A COMPARISON OF FOUR METHODS FOR SOLVING SYSTEMS OF NONLINEAR EQUATIONS+

Michel Yves Cosnard

TR 75-248

Department of Computer Science
Cornell University
Ithaca, NY 14853

+This research was supported in part by the National Science Foundation under Grant GJ-40903.

Abstract

In this thesis Brent's method and the Brent-Gay version of Brown's method for solving systems of nonlinear equations are studied. A comparison is then made between these two methods, Powell's hybrid method and Brown's original method. The numerical results show that the Brent-Gay version is an improvement of the method and that in the average Brent's method is the best. Finally listings of the two studied algorithms and of the test functions are given in the appendices.

Biographical Sketch

Michel Y. COSNARD, born in Grenoble (FRANCE) on July 9, 1952, pursued his studies in Lycée Champollion, Grenoble and passed his Baccalauréat in 1970. The next year he entered the Classes de Mathématiques Supérieures. In July 1972 he succeeded second to the Concours National d'Entrée aux Ecoles Nationales Supérieures d'Ingénieurs. In September he entered the Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble, from which he graduated in June 1975. During the academic year 1974-1975 he was elected as participant in the exchange program between the Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble and Cornell University.

A LISE

Acknowledgements

I would first of all like to express my gratitude to Professor Jorge J. Moré, my thesis advisor, for his excellent guidance and encouragements.

I am indebted to Professor J. E. Dennis, Jr., for serving on my special committee.

I would also like to thank the Institut de Recherche en Informatique et Automatique for supporting my studies at Cornell during the academic year 1974-1975.

Thanks are due to the Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble and to Cornell University for having elected me as a participant in their exchange program.

Finally I would like to thank Miss Sharon Gunkel for her excellent service in typing this thesis.

Table of Contents

Page

PART B - NUMERICAL RESULTS

Part A - THEORY

1. INTRODUCTION

We are interested in the numerical behavior of some methods of solving systems of n nonlinear equations in n unknowns. Such systems can vary widely in size from $n = 2$ up to $n = 1000$ or more. Since the solution is generally not accessible directly it is necessary to search for an approximation by the use of iterative techniques. More precisely, given a mapping $F: \mathbb{R}^n \to \mathbb{R}^n$ from Euclidean n-space into itself we seek a point $x \in \mathbb{R}^n$ at which $||F(x)||$ is suitably small for some norm. Given an initial guess $x_0 \in \mathbb{R}^n$ an iteration constructs a sequence $x_1, x_2, \ldots, x_n, \ldots$ which converges under suitable assumptions to a solution.

1.1 Newton's Method

Assume that

(1.1)
1. (1) F is continuously differentiable
2. (2) There exists $x^* \in \mathbb{R}^n$ such that $F(x^*) = 0$
3. (3) The Jacobian matrix $F'(x^*)$ of F at x^* is nonsingular.

Given $x_0 \in \mathbb{R}^n$ Newton's method computes a linearization of F

$$F(x) \doteq F(x_0) + F'(x_0)(x-x_0)$$

and searches for the point x_1 at which this linearization vanishes

$$F(x_0) + F'(x_0)(x-x_0) = 0$$

(1.2) $$x_1 = x_0 - [F'(x_0)]^{-1} F(x_0)$$

Assume that x_1 is a better approximation of x^* than x_0 and repeat the process:

(1.3) $$x_{i+1} = x_i - [F'(x_i)]^{-1} F(x_i)$$

To compute the next iterate Newton's method does not invert the Jacobian matrix but solves the linear system

(1.4) $$\begin{cases} F'(x_i)s_i = -F(x_i) \\ x_{i+1} = x_i + s_i \end{cases}$$

A justification of the method is the following theorem [14].

Theorem 1.1

Assume that F satisfies (1.1) in an open neighborhood of x^* and that F' satisfies a Lipschitz condition at x^*; that is

(1.5) $$||F'(x)-F'(x^*)|| \leq \alpha \; ||x-x^*|| \quad x \in S \;.$$

then

(1) the iteration (1.4) is locally convergent at x^*

(2) the convergence is Q-quadratic:

$\exists \; c \in \mathbb{R}$ independent of x_i such that

$$||x_{i+1} - x^*|| \leq c \; ||x_i-x^*||^2 \quad \forall i \geq 0$$

1.2 Discretized Newton's Method

The above theorem is stated for exact analytic derivatives.

However in practice it is often difficult and sometimes impossible to obtain the exact Jacobian. We can avoid this difficulty by the use of finite difference approximations. Two typical and frequently used difference approximations are:

$$\partial_j f_i(x) = \frac{1}{2h_{ij}} [f_i(x + h_{ij} e_j) - f_i(x - h_{ij}e_j)]$$

and

$$\partial_j f_i(x) = \frac{1}{h_{ij}} [f_i(x + h_{ij} e_j) - f_i(x)]$$

where the h_{ij} are given discretization parameters and e_j is the j^{th} coordinate vector. Let us call $J(x,h)$ the so obtained approximation to the Jacobian and suppose that

$$\lim_{h \to \infty} J(x,h) = F'(x)$$

then the process

$$x_{i+1} = x_i - [J(x_i,h_i)]^{-1} F(x_i)$$

is called a discretized Newton iteration.

Theorem (1.1) remains true for the discretized Newton iteration but the order of convergence is modified. If h_i is held constant, but sufficiently small, throughout the iteration the method will have only a linear rate of convergence. However if the h_i satisfy

$$\lim_{i \to \infty} h_i = 0$$

then the method will approximate the rapid rate of convergence of Newton's method (for a more detailed discussion of the rate of convergence see [14]). A particularly good choice is obtained when h is function of the iterates.

A disadvantage of Newton's method in its discretized form is that it needs a lot of function evaluations. In practice function evaluations can require considerably more time than all the other calculations. In general Newton's method needs at least $n^2 + n$ component function evaluations per iteration.

Moreover the storage locations and arithmetic operations used in this method are $n^2 + 0(n)$ and $0(n^3)$ where $0(n^k)$ means a multiple of n^k.

The main purpose of the methods we shall study in Section 2, Brown's and Brent's methods, is to decrease these two indices of the efficiency of an algorithm without decreaing too much the rate of convergence.

Last, but not least, another disadvantage of Newton's method is that it enjoys only local convergence. We shall study a way and a method due to Powell to avoid this disadvantage:

1.3 Powell's hybrid method

In 1965 Broyden [4] proposed the following iterative method

$$(1.6)\qquad \begin{cases} x_{i+1} = x_i - B_i^{-1} F(x_i) \\[2ex] B_{i+1} = B_i + \dfrac{(y_i - B_i s_i)s_i^T}{s_i^T s_i} \end{cases}$$

where $y_i = F(x_{i+1}) - F(x_i)$ and $s_i = x_{i+1} - x_i$. Broyden [4] derived the sequence of matrices B_i in such a way that they constitute better approximations to the Jacobian as i increases. Broyden, Dennis and Moré [6] proved that under suitable assumptions Broyden's method is locally and superlinearly convergent. Recall that x_i converges superlinearly to x^* if and only if:

$$\lim_{i\to\infty} \frac{||x_{i+1}-x^*||}{||x_i-x^*||} = 0$$

We can see that this method avoids one disadvantage of Newton's method, i.e., the computation of the Jacobian, since B_i can be updated with only n component function evaluations. Moreover x_{i+1} can be computed in $O(n^2)$ arithmetic operations against $O(n^3)$ for Newton's method.

Powell [15] modified Broyden's method in order to achieve global convergence. This is done by an iteration of the form $x_{k+1} = x_k + p_k$ where p_k is a convex combination of the Broyden correction $- B_k^{-1}\ F(x_k)$ and a multiple of the approximate gradient direction $B_k^T\ F(x_k)$.

In this case the matrices B_k are generated by

$$B_{k+1} = B_k + \theta_k \frac{\left[F(x_k+p_k) - F(x_k) - B_k p_k\right] p_k^T}{p_k^T\ p_k}$$

where the parameter θ_k is chosen so as to avoid singularity in B_{k+1}. We will not describe Powell's method in any detail; the interested reader can find such a discussion in Powell [15] or Moré and Trangenstein [13]. However we would like to state the convergence theorem of Powell [15].

<u>Theorem 1.2</u> Let $F: R^n \to R^n$ be continuously differentiable and assume that F' is bounded on an open set which contains $\left\{x \in R^n : ||F(x)|| \leq ||F(x_0)||\right\}$. Then for each $\epsilon > 0$ Powell's hybrid method produces an index k such that $||B_k^T F(x_k)|| < \epsilon$.

The rate of convergence was analyzed by Moré and Trangenstein [13].

<u>Theorem 1.3</u> Let $F: \mathbb{R}^n \to \mathbb{R}^n$ satisfy the assumptions of Theorem 1.2 and assume that the sequence $\{x_k\}$ generated by Powell's hybrid method converges to a point x^* at which $F'(x^*)$ is nonsingular. Then

(1) $F(x^*) = 0$

(2) $\{x_k\}$ converges R-superlinearly to x^*.

1.4 <u>The interest of a comparison</u>

At this point we have presented two classes of methods for solving systems of nonlinear equations. The first, based upon modification of the discretized Newton's method, achieves <u>local and quadratic convergence</u> in $0(n^2)$ component function evaluations, the second, based upon Broyden's method, achieves <u>global and superlinear convergence</u> in $0(n)$ component function evaluations. However, although these methods have been throughly studied during the past ten years there have not been any extensive numerical results.

For an eventual user it must be interesting to know a priori what are the best methods to solve his problem without studying carefully all the algorithms and theorems.

In fact he can be interested in using a method needing a small amount of storage locations if his problem is of large size while, on the other hand, the stability of the method can be his principal requirement if the Jacobian is very ill-conditioned near the presumed solution.

We do not pretend to answer all possible questions or prove that one method is always better than the others. We only want to show the behavior of the most popular techniques confronted with different kinds of functions. Moreover we would like to study a new class of methods based upon orthogonal transformations and present a way to implement two of them.

3. BRENT'S METHOD

Brent's method is essentially a modified Newton's method based upon orthogonal transformations. It approximates the triangularization of the Jacobian by working one row at a time, eliminating one variable for each row treated [1].

An iteration of that method consists in the construction of a sequence of minor iterates:

$$y_0 = x_i,\ y_1, \ldots, y_{n-1},\ x_{i+1} = y_n$$

such that each y_k is a zero of a linearization of the component function f_j for $j \leq k$. If we call Δ_k the approximation to the gradient of the component function then

$$(2.1) \qquad F_k(y) = f_k(y_{k-1}) + \Delta_k^T(y-y_{k-1})$$

Note that $F_k(y)$ vanishes on an hyperplane H_k normal to Δ_k. Thus y_k is chosen to be a point of $\bigcap_{j=1}^{k} H_j$ since we require that each F_j vanishes at y_k.

The purpose of this chapter is to study how the method works first on a linear system and afterwards, on a nonlinear system. This will lead us to an efficient implementation of the method.

2.1 Linear system

To understand the method more clearly we show how it works on a linear system $Ax = b$ where A is nonsingular.

1st stage

We have

$$A = \begin{bmatrix} x & \cdots\cdots & x \\ x & & \vdots \\ \vdots & & \vdots \\ x & \cdots\cdots & x \end{bmatrix} \qquad y_1 = \begin{bmatrix} x \\ \vdots \\ x \end{bmatrix}, \; Q_1$$

where Q_1 is an orthogonal matrix of the order of A.

kth stage

At the beginning of the kth stage we have Q_k, AQ_k and y_k

$$(2.2) \quad AQ_k = \begin{array}{cc} & \overbrace{\qquad\qquad\qquad}^{k-1} \\ k-1\left\{\begin{array}{c} \\ \\ \\ \\ \end{array}\right. & \left[\begin{array}{ccc|c|ccc} x & & 0 & & & 0 & \\ \vdots & \ddots & & & & & \\ \vdots & & \ddots & & & & \\ \vdots & & & x & & & \\ \hline x & \cdots & \cdots & x & \cdots & \cdots & x \\ \hline \vdots & & & \vdots & & & \vdots \\ x & \cdots & \cdots & x & \cdots & \cdots & x \end{array}\right] \end{array}$$

Let us call w_k the k^{th} row of AQ_k without its (k-1) first elements:

$$w_k = \begin{bmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ x \\ \cdot \\ \cdot \\ \cdot \\ x \end{bmatrix} .$$

Next we compute an orthogonal P_k of the form

$$P_k = \left[\begin{array}{c:c} I_{k-1} & 0 \\ \hdashline 0 & \hat{P}_k \end{array}\right]$$

where I_{k-1} is the identity matrix of order k-1 and such that:

$$(2.3) \quad P_k^T w_k = \alpha_k e_k \qquad \alpha_k = \pm ||w_k||_2$$

If we now set

$$(2.4) \quad Q_{k+1} = Q_k P_k ,$$

it is clear that AQ_{k+1} is of the required form. Finally we define

$$(2.5) \quad y_{k+1} = y_k + \frac{(b - Ay_k)}{\alpha_k} Q_{k+1} e_k .$$

nth stage

At the end of the nth stage we have an orthogonal matrix Q_{n+1} such that $AQ_{n+1} = L$ where L is a lower triangular matrix. The solution is in y_{n+1}.

To show that this works we prove the following theorem:

Theorem 2.1 Brent's method solves the linear system $Ax = b$ where $b \in \mathbb{R}^n$ and $A \in L(\mathbb{R}^n, \mathbb{R}^n)$ is nonsingular in one iteration.

Proof: We use an induction proof. Let us assume

$$(2.6) \qquad Ay_k = \begin{bmatrix} b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_{k-1} \\ x \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x \end{bmatrix}$$

and show that

$$(2.7) \qquad Ay_{k+1} = \begin{bmatrix} b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_{k-1} \\ b_k \\ x \\ \cdot \\ \cdot \\ \cdot \\ x \end{bmatrix}$$

From (2.5) we obtain

$$\text{(2.8)}\quad Ay_{k+1} = Ay_k + \frac{e_k^T(b - Ay_k)}{\alpha_k}\, AQ_{k+1}\, e_k$$

Multiply by e_j^T and apply the induction hypothesis (2.6):

$$e_j^T Ay_{k+1} = e_j^T Ay_k = b_j \qquad 1 \le j \le k-1$$

Now we prove the same result for $j = k$

$$\text{(2.9)}\quad e_k^T Ay_{k+1} = e_k^T Ay_k + \frac{e_k^T (b - Ay_k)}{\alpha_k}\, e_k^T AQ_{k+1}\, e_k$$

From (2.2), (2.3) and (2.4) we know that

$$\text{(2.10)}\quad e_k^T AQ_{k+1}\, e_k = e_k^T AQ_k\, P_k\, e_k = a_{k.}\, P_k\, e_k = \alpha_k$$

where a_k is the kth row of AQ_k.

Applying (2.10) in (2.9) we get

$$e_k^T Ay_{k+1} = e_k^T Ay_k + \frac{e_k^T (b - Ay_k)}{\alpha_k}\, \alpha_k = e_k^T b = b_k$$

which proves (2.7) and the theorem follows.

Note that this process is related to the QL algorithm for linear systems but this implementation is less efficient since it needs to compute and keep the full orthogonal matrix Q.

2.2 Nonlinear System

The generalization is straightforward to the case where F is nonlinear. Let us describe the subiteration loop.

1st stage

We have an orthogonal matrix Q_1 (at the beginning of the algorithm we may choose $Q_0 = I$), an iterate x_i and a subiterate y_1 such that:

$$y_1 = x_i$$

(k+1)th stage

At the end of the kth stage we have Q_k, y_k. We compute w_k:

$$w_k = \frac{1}{h_i} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ f_k(y_k+h_i Q_k e_k) - f_k(y_k) \\ \vdots \\ f_k(y_k+h_i Q_k e_n) - f_k(y_k) \end{bmatrix}$$

Note that w_k can be considered as the kth row of an approximation of the Jacobian matrix without its (k-1) first elements.

Now we compute an orthogonal P_k such that

$$P_k^T w_k = \alpha_k e_k \qquad \alpha_k = \pm ||w_k||_2$$

P_k can be chosen as a product of plane rotations or as a Householder transformations (see 5.1.2).

Then we get

$$Q_{k+1} = Q_k P_k$$

$$Y_{k+1} = Y_k - \frac{f_k(Y_k)}{\alpha_k} Q_{k+1} e_k$$

<u>nth stage</u>

At the end of the nth stage we have an orthogonal matrix Q_{n+1} and a vector Y_{n+1} which is considered as a better approximation to a zero of F than x_i. We then set $x_{i+1} = y_{n+1}$ and start if necessary a new iteration.

Thus the method can be written in the following simple form:

<u>beg</u>: <u>if</u> $F(x_i) = 0$ <u>then</u> stop

$y_1 = x_i$

For $k = 1, n$

compute w_k and P_k

$Q_{k+1} = Q_k P_k$

$y_{k+1} = y_k - \frac{f_k(y_k)}{\alpha_k} Q_{k+1} e_k$

$x_{i+1} = y_{n+1}$

<u>go to</u> beg

In his original paper [1] Brent stated the following theorem which shows that under suitable assumptions the method enjoys <u>local quadratic convergence</u>.

<u>Theorem 2.2</u> Suppose that F satisfies (1.1) and (1.5) in an open neighborhood $S = \left\{ x \in R^n : ||x-x^*|| < \varepsilon \right\}$. If $||h_i|| = O(||F(x_i)||)$ and if $||x_0-x^*||$ is sufficiently small

then the previous algorithm generates a sequence $\{x_i\}$ which converges quadratically to x^*.

3. BROWN'S METHOD

Brown's method is a variation of the discretized Newton's method based upon Gaussian elimination with partial pivoting in such a way that the most recent information is always used at each step of the algorithm. We shall describe in Section 3.1 the method in its initial form as stated by Brown [3]. Then we shall show how to implement Brown's method in another way. This modified version was first suggested by Brent [1] and then followed up and generalized by David Gay [8].

3.1 The initial version

The method approximates the forward triangularization of the Jacobian matrix by working with one row at a time, eliminating one variable for each row treated. Let us describe the ith iteration of the algorithm.

1st stage

As in Section 2, Δ_1 denotes the approximation to the gradient of the first component function then

$$F_1(x) = f_1(x_i) + \Delta_1^T(x - x_i) \tag{3.1}$$

can be considered as a good approximation to $f_1(x)$ in a neighborhood of x_i. Set $F_1(x)$ to zero and solve the linear equation for the variable whose corresponding partial derivative is largest in absolute value (this corresponds to the partial row pivoting of Gaussian elimination). Assume that this variable corresponds to the n-th component ξ^n of x. Note that if $x = (\xi^1,\ldots,\xi^n)$ then ξ^n is a function of $\xi^1,\ldots,\xi^{n-1}$.

<u>2nd stage</u>

Define a function g_2 of the (n-1) variables $\xi^1,\ldots,\xi^{n-1}$ by

$$(3.2) \qquad g_2(\xi^1,\ldots,\xi^{n-1}) = f_2(\xi^1,\ldots,\xi^{n-1}, \xi^n).$$

Repeat the same process with g_2 instead of f_2 as in the first stage to eliminate one more variable.

<u>nth stage</u>

At this stage we have

$$(3.3) \qquad g_n(\xi^1) = f_n(\xi^1,\ldots,\xi^n)$$

where $\xi^2,\ldots,\xi^n$ are functions of the variable ξ^1 and can be obtained by back substitution. We linearize $g_n(\xi^1)$, set the linearization to zero and solve the linear equation in one unknown ξ^1. We consider that ξ^1 is a better approximation to the first component of x^* than the first component of x_i. Then using ξ^1 and $g_{n-1}(\xi^1,\xi^2)$ we back solve to get ξ^2. Eventually we shall get a new vector $(\xi^1,\ldots,\xi^n)$ which we consider to be the new iterate x_{i+1}.

Brown and Dennis [2] showed that, like Newton's method, Brown's method enjoys <u>local quadratic convergence</u>. Moreover a great advantage of the method is that it requires almost half as many function evaluations as a discrete Newton's method. However the amount of work per iteration measured by the number of arithmetic operations is increased and now becomes $O(n^4)$. We shall describe a new way to implement Brown's method in order to avoid this disadvantage.

3.2 A modification of Brown's method

The idea suggested by Brent [1] and followed by Gay [8] is that if we replace in Brent's algorithm the matrix P_k by a matrix E_k, no longer orthogonal but such that $Q_k E_k$ is triangular, we get a new method based upon Gaussian elimination [8].

Let us describe briefly how the method works for a linear system. The development is very similar to that in Section 2.1.

Consider the linear system $Ax = b$ where A is non-singular.

1st stage

We have A, y_1 and $R_1 = I$ where I is the identity matrix of the order of A.

kth stage

At the beginning of the kth stage we have R_k, AR_k and y_k as in Section 2.1 with R_k replacing Q_k. Then we compute Π_k and E_k with Π_k a permutation matrix (corresponds to the partial pivoting) and

(3.4) $E_k = I - e_k m_k^T$

(3.5) $E_k^T \Pi_k W_k = \beta_k e_k$

where m_k has its first k-1 components equal to zero and w_k is defined in Section 2.1.

We then get:

(3.6) $A R_k \Pi_k E_k = AR_{k+1}$

(3.7) $y_{k+1} = y_k + \dfrac{e_k^T (b - Ay_k)}{\beta_k} R_{k+1} e_k$

<u>nth stage</u>

At the end of the nth stage we obtain R_{n+1} and L such that $AR_{n+1} = L$ with L lower triangular. As in Section 2.1 we state the following theorem.

<u>Theorem 3.1</u>

The Brent-Gay version of Brown's method solves the linear system $Ax = b$ where $b \in R^n$ and $A \in L(\mathbb{R}^n, \mathbb{R}^n)$ is non-singular in one iteration.

<u>Proof</u>: The proof is obviously similar to that of Theorem 2.1. So we omit it.

We can generalize as in Section 2.1 to the case of a nonlinear system.

<u>1st stage</u> $R_1 = I$ $\quad Y_1 = X$

(k+1)th stage

At the end of kth stage we have R_k and y_k. We compute:

$$W_k = \frac{1}{h_i} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ f_k(y_k+h_i R_k e_k) - f_k(y_k) \\ \vdots \\ f_k(y_k+h_i R_k e_n) - f_k(y_k) \end{bmatrix}$$

Find Π_k and E_k as described in (3.4) and (3.5) such that

(3.8) $E_k^T \, \Pi_k \, W_k = \beta_k \, e_k^T$

then compute

(3.9) $R_{k+1} = R_k \, \Pi_k \, E_k$

(3.10) $y_{k+1} = y_k - \dfrac{f_k(y_k)}{\beta_k} R_{k+1} \, e_k$

So the method can be written in the following simple form:

<u>beg</u> : <u>if</u> $F(x_i) = 0$ <u>then</u> stop
$y_1 = x_i$
$R_1 = I$
<u>For</u> $k = 1,n$
  compute W_k, Π_k and E_k
  $R_{k+1} = R_k \, \Pi_k \, E_k$
  $y_{k+1} = y_k - \dfrac{f_k(y_k)}{\beta_k} R_{k+1} \, e_k$
$x_{i+1} = y_{n+1}$
<u>go to</u> beg

We are not interested in showing the equivalence of the two versions since it is not an easy work. We refer the reader to Gay [8]. We shall show in the next section how to implement this version of Brown's method. For this purpose the following theorem will be very useful.

<u>Theorem 3.2</u> The product $\Pi_k \ldots \Pi_1 R_{k+1}$ is a unit upper triangular matrix of the form:

$$\Pi_k \ldots \Pi_1 R_{k+1} = \left[\begin{array}{c|c} L_k & S_k \\ \hline 0 & I_{n-k} \end{array}\right]$$

where L_k is a unit upper triangular matrix of order k and I_{n-k} the identity matrix of order n-k.

<u>Proof</u> The proof is by induction. Assume that

$$\Pi_{k-1} \ldots \Pi_1 R_k = \left[\begin{array}{c|c} L_{k-1} & W \\ \hline 0 & I_{n-k+1} \end{array}\right]$$

where L_{k-1} is a unit upper triangular matrix of order k-1. Then

$$\Pi_k \ldots \Pi_1 R_k \Pi_k = \left[\begin{array}{c|c} L_{k-1} & \tilde{W} \\ \hline 0 & I_{n-k+1} \end{array}\right]$$

Since $E_k = I - e_k m_k^T$ then

$$\Pi_k \ldots \Pi_1 R_{k+1} = \Pi_k \ldots \Pi_1 R_k \Pi_k - V_k m_k^T$$

where

$$V_k = \Pi_k \cdots \Pi_1 R_k \Pi_k e_k$$

Since V_k has zeroes in the last (n-k) positions and m_k has zeroes in its first k positions it follows that $\Pi_k \cdots \Pi_1 R_{k+1}$ is of the required form.

For linear systems this method is related to the LU decomposition with partial pivoting columnwise. We do not know if there is any advantage to this version.

Brown and Dennis [2] have proved the following theorem which shows that under suitable assumptions the method enjoys local quadratic convergence.

Theorem 3.3 Suppose that F satisfies (1.1) and (1.5) in an open neighborhood $S = \left\{ x \in \mathbb{R}^n : ||x-x^*|| < \varepsilon \right\}$. If $||h_i|| = O(||F(x_i)||)$ and if $||x_0-x^*||$ is sufficiently small then the previous algorithm generates a sequence $\{x_i\}$ which converges quadratically to x^*.

4. COMPARISON OF THE FOUR METHODS

In the first three sections of this thesis we have studied four methods of solving systems of nonlinear equations. Powell's hybrid method, Brent's method and two versions of Brown's method. Our purpose in this section is to study the properties of these methods in order to compare each with the others. Our comparison will be on the following criteria: convergence, function evaluations, storage and number of arithmetic operations requirements.

4.1 Convergence

We have seen in the first section that Powell's method shows a global and superlinear rate of convergence. Recall that $\{x_k\}$ converges superlinearly to x^* if and only if:

$$\lim_{k\to\infty} \frac{||x_{k+1} - x^*||}{||x_k - x^*||} = 0$$

On the other hand, Brent's method and Brown's method in its two versions enjoy local but quadratic convergence. Recall that $\{x_k\}$ converges quadratically to x^* if and only if:

$$||x_{k+1} - x^*|| \le c||x_k - x^*||^2, \; k \ge 0$$

However we have to attenuate the differences between the two classes of methods. In fact, the theorems presented in the first sections require all the assumptions (1.1) and (1.5). Moreover Powell's hybrid method also requires a condition of <u>uniform linear independence</u> upon the directions generated by the algorithm (for a discussion of this condition see

Moré and Trangenstein [13]) but is then globally convergent.

So the methods will converge with the appropriate rate of convergence only for regular enough functions. For many of these functions the neighborhood in which the local convergence is assured will be big enough to conclude that the difference between the two classes of methods is not verified. On the other hand, near the solution Powell's hybrid method will reduce to Broyden's method and therefore converges superlinearly.

This is why for a great number of functions (for example if we have a good guess) the computational requirements will be more important than the convergence characteristics.

4.2 Computational Requirements

According to Brown and Powell their methods require respectively:

$\frac{n^2 + 3n}{2}$ component function evaluations, $0(n^4)$ arithmetic operations and $\frac{n^2}{2} + 0(n)$ storage locations per iteration for the initial version of Brown's method.

n component function evaluations, $0(n^2)$ arithmetic operations and $3n^2 + 0(n)$ storage locations per iteration for Powell's hybrid method.

In order to compare the four methods we show two properties of Brent's method and the modified version of Brown's method.

Property 4.1

Bront's method costs per iteration

$\frac{n^2+3n}{2}$ component function evaluations

$n^3 + 0(n^2)$ real multiplications

$\frac{3}{2} n^3 + 0(n^2)$ real additions

$n^2 + 0(n)$ storage locations

Proof Since each vector w_k needs $(n-k+2)$ component function evaluations we find:

$$(4.1) \quad N_f = \sum_{k=1}^{n} (n-k+2) = n^2 - \frac{(n+1)n}{2} + 2n = \frac{n^2+3n}{2}$$

The bulk of the work concerning the real multiplications is the computation of Q_{k+1}. Since P_k can be chosen to be a Householder transformation (for the choice of P_k see 5.1) of the form

$$P_k = I - \Pi^{-1} w_k w_k^T$$

where w_k has zeroes in its first k positions, we can compute Q_{k+1} by

$$Q_{k+1} = Q_k - \Pi^{-1} Q_k w_k w_k^T$$

The computation of Q_{k+1} then requires $\Pi^{-1} Q_k w_k = v_k$ at a cost of $n(n-k)$ multiplications and then $v_k w_k^T$ which costs an additional $n(n-k)$ multiplications. In all the number of multiplications is

(4.2) $N_m = 2 \sum_{k=1}^{n} n(n-k) = n^3 + 0(n^2)$

The computation of the Q_k's requires also $n^3 + 0(n^2)$ real additions but there is an additional term coming from the computation of w_k. In fact we have to compute

$$y_k + h_i Q_k e_j \qquad\qquad k \le j \le n \quad .$$

We can avoid easily (see Section 5.1) the multiplication by h_i but not the addition which introduces $\frac{n^3}{2}$ additions.

(4.3) $N_A = n^3 + \sum_{k=1}^{n} n(n-k) = \frac{3n^3}{2} + 0(n^2)$

If we consider the storage requirement the algorithm needs only the storage of the matrix Q_k of size n^2 and of some vectors of size n which proves the property.

<u>Property 4.2</u>

Brown's method in its modified version costs per iteration

$\frac{n^2+3n}{2}$ component function evaluations

$\frac{1}{6}n^3 + 0(n^2)$ real multiplications

$\frac{1}{3}n^3 + 0(n^2)$ real additions

$\frac{n^2}{4} + 0(n)$ storage locations

<u>Proof</u> If we consider the computation of w_k in Brent's method and Brown's method it is obvious that they are the same.

The result follows:

(4.4) $N_f = \frac{n^2 + 3n}{2}$

Let us call $u_1^{(k)}, \ldots, u_n^{(k)}$ the column vectors of R_k. At the kth stage the vectors $u_1^{(k)}, \ldots, u_{k-1}^{(k)}$ are no longer needed, hence the algorithm requires $\frac{n^2}{4}$ storage locations for the nontrivial parts of $u_k^{(k)}, \ldots, u_n^{(k)}$ since $u_j^{(k)}$ differs from

$$\begin{pmatrix} \bar{u}_j^{(k)} \\ \hline e_{j-k+1} \end{pmatrix}$$

(with e_j the jth column vector of I_{n-k+1}) by a permutation matrix. Indeed the size of the needed storage is:

$$N_s = \max[k(n-k)]$$

This maximum is obtained when the two numbers are equal. Hence if $k = n-k$ then $k = n/2$ and thus

$$(4.5) \qquad N_s = \begin{bmatrix} \frac{n^2}{4} & \text{if } n \text{ even} \\ \frac{n^2-1}{4} & \text{if } n \text{ odd} \end{bmatrix}$$

Since we need $O(n)$ storage locations for the working arrays the result is proved.

To compute the u_j's we use $k(n-k)$ real additions and multiplications per subiteration. To compute the w_k's we use $k(n-k)$ real additions:

$$N_m = \sum_{k=1}^{n} k(n-k) = \frac{1}{6}n^3 + O(n^2)$$

$$N_A = 2 \sum_{k=1}^{n} k(n-k) = \frac{1}{3}n^3 + 0(n^2)$$

For a more detailed discussion of the implementation of Brent's and Brown's method we refer the reader to the Section 5.

We have summarized the properties of the methods in the following array:

METHOD	CONVERGENCE	COMP. FUNC. EVAL.	STORAGE	OPERATIONS
BROWN	LOCAL QUADRATIC	$\frac{n^2+3n}{2}$	$\frac{n^2}{2} + 0(n)$	$0(n^4)$
POWELL	GLOBAL SUPERLINEAR	n	$3n^2 + 0(n)$	$0(n^2)$
BRENT	LOCAL QUADRATIC	$\frac{n^2+3n}{2}$	$n^2 + 0(n)$	$n^3 + 0(n^2)$ mults. $\frac{3}{2}n^3 + 0(n^2)$ adds.
MODIFIED VERSION	LOCAL QUADRATIC	$\frac{n^2+3n}{2}$	$\frac{n^2}{4} + 0(n)$	$\frac{1}{6}n^3 + 0(n^2)$ mults. $\frac{1}{3}n^3 + 0(n^2)$ adds.

The array suggests principally three conclusions:

- Brown's method seems to be advantagoously replaced by the modified version

- Brent's method seems to show an increase of more than 100% in the computational expenses (as in QR versus

Gaussian elimination) but we shall see that Brent's version seems to be more stable.

- If n iterations of Powell's method correspond to more than two of the others, Powell's hybrid method is computationally more efficient than the others.

From this study we can see the importance of a numerical comparison since each method has advantages and disadvantages and none seems better than the others.

Part B - NUMERICAL RESULTS

5. IMPLEMENTATION OF BRENT'S AND BROWN'S METHODS

Brent's method and the modified version of Brown's method are recent. So it seems interesting to give a way to implement them efficiently. Moreover some problems arise in order to perform the methods optimally as for instance, the choice of P_k in Brent's method, or the storage of the u_j's in Brown's method.

5.1 Efficient implementation of Brent's method

As described in Section 3 the algorithm is the following:

<u>Comment</u> stopping criterion

top: <u>If</u> $f(x_{i-1}) = 0$ <u>then</u> stop $x_j = x_{i-1}$ $\forall j \geq i$

<u>Comment</u> initialization

- Q is the orthogonal matrix
- h is the step for the finite difference formula

Choose h

$Q_1 = Q_{n+1}$

$y(j) = x(j)$, $j = 1 \to n$

<u>Comment</u> enter the subloop.

<u>For</u> $k = 1 \to n$

$w(j) = \frac{1}{h}[f_k(y+h\ Q_k\ e_j)-f_k(y)], j = k \to n$

compute P_k such that $P_k w = \alpha\ e_k$

$$
\left\lfloor \begin{array}{l} Q_{k+1} = Q_k\, P_k \\ y(j) = y(j) - \dfrac{f_k(y)}{\alpha}\, e_j^T\, Q_{k+1}\, e_k, j = 1 \to n \end{array} \right.
$$

$x(j) = y(j)\ ,\ j = 1 \to n$

$i = i+1$

<u>go to</u> top.

First we can remark that the multiplication by h in the term $y + h\, Q_k\, e_j$ can be avoided. For this we introduce h in Q_k before computing w and take it out after. Moreover in the previous algorithm we are confronted with the problem of finding an orthogonal matrix P_k such that, given a vector z_k we have:

$$(5.1)\quad P_k\, z_k = \alpha\, e_k$$

This reduction can be done in two ways; by the use of a product of plane rotations or a Householder transformation. We shall study both ways.

<u>Product of plane rotations</u>

A plane rotation in the plane (i,j) is a matrix of the form:

$$
\begin{array}{rcl}
 & \begin{array}{ccccccc} & & & \text{ith column} & & & \\ & & & \downarrow & \text{jth column} & & \\ & & & & \downarrow & & \end{array} & \\
P_{ij} = & \left[\begin{array}{ccccccc} 1 & & & & & & \\ & \ddots & & & 0 & & \\ & & 1 & & & & \\ & & & \gamma_1 & \sigma & & \\ & 0 & & -\sigma \ \ddots & 1\ \gamma_1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{array}\right] & \begin{array}{l} \\ \\ \\ \leftarrow \text{ith row} \\ \leftarrow \text{jth row} \\ \\ \\ \end{array}
\end{array}
$$

where $\gamma^2 + \sigma^2 = 1$. Clearly P_{ij} can be obtained by embedding the matrix $\begin{bmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{bmatrix}$ in the identity matrix of order n [17].

Given a vector z, by choosing γ and σ in an appropriate way, the multiplication of z by P_{ij} introduces a zero in the jth position of z.

$$\delta^2 = z_i^2 + z_j^2$$

$$\delta = \mathrm{sign}(z_i)\sqrt{\delta^2}$$

$$\gamma = \frac{z_i}{\delta} \qquad \sigma = \frac{z_j}{\delta}$$

Obviously the sequence of plane rotations $P_{k,k+1},\ldots,P_{n-1,n}$ will introduce zeroes in the $n,(n-1),\ldots,(k+1)$th positions of z and will perform the desired reduction. The product of Q by P_{ij} seems to require 4n multiplications and 2n additions so that the complete reduction will cost 4n(n-k) multiplications and 2n(n-k) additions plus (n-k) square roots which is a big price. Gill, Golub, Murray and Saunders [10], Gentleman [9] and Hanson [12] studied some ways to reduce the overhead up to 2n(n-k) multiplications and 2n(n-k) additions but these computations can lead easily to overflow and underflow and are very sensitive to round off errors.

A product $P_{i,i+1},\ldots,P_{j,j+1}$ of plane rotations is an upper (or lower) Hessenberg matrix [10] and can be stored in $\frac{n^2}{2} + O(n)$ storage locations. Made in the regular form the product of Q_k by $P_{i,i+1}$ costs $4(n+k-i-1) + 2$ multiplications. So the computation of Q_{n+1} requires:

$$N = \sum_{k=1}^{n} \sum_{k=1}^{n} 4(n+k-i-1) + 2 = \frac{4}{3} n^3 + O(n^2)$$

real multiplications. However by the use of the two-multiply, two-add Given's rotation [12] we can reduce the overhead to

$$\frac{2n^3}{3} + O(n^2) \text{ real multiplications}$$

$$\frac{2n^3}{3} + O(n^2) \text{ real additions}$$

$$\frac{n^2}{2} + O(n) \text{ storage locations}$$

at a price that Q_{n+1} is not entirely accessible so that we have to start each new iteration with the identity matrix.

Gay [8] showed by the use of Powell's orthogonalization scheme another way to reduce the overhead which seems more stable.

Householder transformations

A Householder transformation is a matrix of the form $P = I - \Pi^{-1} u\, u^T$ where

$$u = \begin{bmatrix} \xi_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \xi_n \end{bmatrix} \qquad \begin{aligned} s^2 &= \xi_1^2 + \ldots + \xi_n^2 \\ s &= \mathrm{sign}(\xi_1)\sqrt{s^2} \\ \Pi &= s(\xi_1 + s) \end{aligned}$$

It is not too difficult to see that [17]

$$Pu = \begin{bmatrix} \xi_1 + s \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix},$$

so that the previous reduction can be done with one Householder transformation. The product of Q_k by P_k can be computed in $2n(n-k)$ real multiplications by the following algorithm

$$Q_k P_k = Q_k - \Pi^{-1} Q_k u u^T$$

(1) Compute $v = \Pi^{-1}\, Q_k\, u$

(2) Compute $Q_k\, P_k = Q_k - v u^T$

so that the computation of Q_{n+1} requires N' multiplications where

$$N' = \sum_{k=1}^{n} 2n(n-k) = 2n\left(n^2 - \frac{n^2}{2}\right) = n^3$$

Moreover since P_1 is a full matrix the necessary storage is n^2 locations but we can start each new iteration by using the orthogonal matrix obtained at the end of the previous iteration.

We can see that the big difference between the two methods is that using plane rotations requires starting each new iteration with the identity matrix. We do not yet know the numerical effect of such a difference, but the theoretical rate of convergence is not affected.

5.2 Efficient Implementation of Brown's Method

In section 3 we described Brown's method in terms of a sequence of n+1 matrices $R_1,\ldots,R_{n+1}$. If we denote the columns of R by $u_1,\ldots,u_n$ then Brown's method is as follows:

<u>Comment</u> stopping criterion

top: <u>If</u> $f(x_{i-1}) = 0$ <u>then</u> stop $x_j = x_{i-1}$ $\forall j \geq i$

<u>Comment</u> initialization

- h is the step for the finite difference formula
- Π is the permutation vector.

Choose h

$y(j) = x(j)$ $\quad j = 1 \to n$

$u_j = e_j$ $\quad j = 1 \to n$

$\Pi(j) = j$ $\quad j = 1 \to n$

__Comment__ enter the subloop

__For__ $k = 1 \to n$

$w(j) = \frac{1}{h}[f_k(y+h\,u_j) - f_k(y)], j = k \to n$

compute j^* such that $w(j^*) = \max\{w(j): k \le j \le n\}$

relabel if necessary

- exchange $w(j^*)$ and $w(k)$
- exchange $u(j^*)$ and $u(k)$
- $\Pi(k) = j^*$ and $\Pi(j^*) = k$

$$y(j) = y(j) - \frac{f_k(y)}{w(k)} u_k(j), j = 1 \to n$$

$$u_\ell(j) = u_\ell(j) - \frac{w(\ell)}{w(k)} u_k(j) \qquad j = 1 \to k$$
$$\ell = k+1 \to n$$

$x(j) = y(j) \quad , \quad j = 1 \to n$

$i = i+1$

__goto__ top.

We now study how to store the vectors u_k in order to use the property 4.2.

__The storage of u_k.__

Property 4.2 shows us that we can store the u_k's in a vector of length $[n^2/4]$ with

$$[n^2/4] = \frac{n^2}{4} \quad \text{if } n \text{ even}$$

$$[n^2/4] = \frac{n^2-1}{4} \text{ if } n \text{ odd.}$$

The u_j's would be of the following form

$$u_j^{(k)} = \begin{bmatrix} \bar{u}_j^{(k)} \\ \hline e_{j-k+1} \end{bmatrix}$$

(with e_j the jth column vector of I_{n-k+1}) if there was no relabel. Indeed because of the partial pivoting the matrix R differs from an upper triangular matrix by a permutation matrix (Theorem 3.2). So at each step we keep in a vector Π the permutation of the rows of R.

Moreover we store R by rows keeping only the non-trivial part of each row. In the following algorithm we describe more precisely how the storage is made.

From theorem 3.2 we know that at the beginning of the kth stage:

$$\Pi_{k-1} \cdots \Pi_1 R_k = \left[\begin{array}{c|c} L_{k-1} & W \\ \hline 0 & I_{n-k+1} \end{array}\right]$$

where W is a $(k-1)$ by $(n-k+1)$ matrix. Since at the kth stage only columns k through n of R_k are needed, we only store W and now show how to update it.

First of all, note that

$$\Pi_k \cdots \Pi_1 R_k \Pi_k = \left[\begin{array}{c|c} L_{k-1} & \tilde{W} \\ \hline 0 & I_{n-k+1} \end{array}\right]$$

where $\tilde{W}$ only differs from W in that the kth and j*th columns are interchanged. If W is stored by rows then this interchange can be done by means of the following algorithm

```
index = j* - (k-1)
ind   = 1
For i = 1 → k-1
    interchange u(index) and u(ind)
    index = index + (n-k+1)
    ind   = ind + (n-k+1)
```

Since $E_k = I - e_k m_k^T$ (where m_k has zeroes in its first k positions) we now have

$$\Pi_k \cdots \Pi_1 R_{k+1} = \Pi_k \cdots \Pi_1 R_k \Pi_k - v_k m_k^T$$

where

$$v_k = \Pi_k \cdots \Pi_1 R_k \Pi_k e_k .$$

Since v_k has zeroes in its last (n-k) positions and m_k has zeroes in its first k positions it follows that

$$\Pi_k \cdots \Pi_1 R_{k+1} = \left[\begin{array}{c:c} L_k & \hat{W} \\ \hdashline 0 & I_{n-k} \end{array}\right]$$

where $\hat{W}$ is now a k by (n-k) matrix. To obtain $\hat{W}$ note that for $k+1 \le j \le n$ and $1 \le i \le k-1$,

$$e_i^T \tilde{W} e_j = e_i^T W e_j - v_k(i)\, m_k(j),$$

and that for $i = k$

$$e_k^T \tilde{W} e_j = - m_k(j).$$

It should now be clear that the following algorithm updates W by rows

```
index = 1
ℓ = 1
For i = 1 → k-1
|   For j = 1 → n-k
|   |   u(ℓ) = u(index+j) - u(index) m_k(k+j)
|   └   ℓ = ℓ + 1
└   index = index + (n-k+1)
For j = k+1 → n
|   u(ℓ) = -m_k(j)
└   ℓ = ℓ+1
```

<u>Theorem 5.1</u>

The previous algorithm solves the storage problem of modified Brown's method.

<u>Proof:</u>

(1) <u>We have enough storage</u>

We need $k(n-k)$ storage locations to store the k vectors of $(n-k)$ components each since the 1 is never stored. $g(k) = k(n-k)$ can be considered as a function of k. It has a maximum for $k = k_{max}$:

$$g'(k) = n-2k \Rightarrow n-2k_{max} = 0 \Rightarrow k_{max} = \frac{n}{2}$$

Since we have to consider k integer:

(a) n even, $k_{max} = \frac{n}{2}$, $g(k_{max}) = \frac{n^2}{4}$.

(b) n odd, $k_{max} = \frac{n-1}{2}$ or $\frac{n+1}{2}$, $g(k_{max}) = \frac{n^2-1}{4}$

(2) <u>We store all the necessary information</u>

(a) To compute w_j we need $z = y + h\, u_j$ $k \le j \le n$ this is made by

```
For j = k → n
    ℓ = j-k+1
    For i = 1 → k-1
        z(Π(i)) = y(Π(i)) + h u [(i-1)(n-k+1)+ℓ]
    z(Π(j)) = y(Π(j))+h
```

The last instruction corresponds to the 1 of u_j that we do not store.

(b) To compute the next subiterate $Y = Y + \alpha\, u_k$, we use the following sequence

```
For i = 1 → k-1
    y(Π(i)) = y(Π(i)) + α u [(i-1)(n-k+1)+1]
y(Π(k)) = y(Π(k)) + α
```

The last instruction corresponds to the 1 of u_j.
More clearly we compute y as:

$$\Pi y = \Pi y + \Pi R_k\, e_k$$

with ΠR_k unit triangular. The matrix ΠR_k is that which we store in u.

We can suppress here also the multiplication by h in the computation of $y + h\, u_j = z$. For this we only have to start with hI instead of I at the initialization of the u_i's which is the same as making all the computations with h included in u_j.

6. DESCRIPTION OF THE TEST FUNCTIONS

To compare numerically the four methods we use two classes of functions:

- three well-known functions: Rosenbrock's function, Powell's singular function, Chebyquad [7].
- four functions derived from two differential equations.

For each of these functions an initial guess is available.

6.1 The Well-Known Functions

Rosenbrock's function: this function is often used in minimization problems (in 1960 Rosenbrock stated the problem of minimizing the 2-norm of the function). It is a function of two variables with a linear first component:

$$f_1(x_1,x_2) = 1-x_1$$
$$f_2(x_1,x_2) = 10(x_2-x_1^2)$$

the initial guess is $(-1.2,1.0)^T$ and the solution is $(1.0,1.0)^T$.

Powell's Singular Function: this function was first defined by Powell in 1962 and is of four variables:

$$f_1(x_1,x_2,x_3,x_4) = x_1+10x_2 \qquad f_2(x_1,x_2,x_3,x_4) = \sqrt{5}(x_3-x_4)$$
$$f_3(x_1,x_2,x_3,x_4) = (x_2-2x_3)^2 \quad f_4(x_1,x_2,x_3,x_4) = \sqrt{10}(x_1-x_4)^2$$

the starting point is $(3,-1,0,1)^T$ and the solution is $(0,0,0,0)^T$. An interesting property of this function is that its Jacobian is singular at the solution so that the hypothesis (1.1) does not hold and the convergence theorems are thus not applicable. So we can expect the rate of convergence to be less than the normal one.

Chebyquad: this class of functions was proposed by Fletcher [7] in 1965.

$$f_i(x_1, \ldots, x_n) = \int_0^1 T_i(x)\,dx - \frac{1}{n}\sum_{j=1}^{n} T_i(x_j) \quad 1 \leq i \leq n$$

where T_i is the ith shifted Chebyshev polynomial and:

$$\int_0^1 T_i(x)\,dx = \begin{cases} 0 & \text{if } i \text{ is odd} \\ -\dfrac{1}{i^2-1} & \text{if } i \text{ is even} \end{cases}$$

the initial guess is given by $x_i^o = \frac{i}{n+1}$. The function has a zero $x^* \doteq (x^*_1, x^*_2, \ldots, x^*_n)$ such that $0 \leq x^*_i \leq 1$ for $n \leq 9$ (except for $n = 8$). However, it is important to note that, given a solution x^*, any permutation of its components gives another solution. We shall see the effects of that property in the next section.

6.2 Differential Equation

It is well known [14] that two point boundary value problems can lead to systems of nonlinear equations. More precisely, given the problem of finding $u(\Gamma)$ such that

(6.1) $\quad u'' = f(\Gamma, u) \qquad 0 \leq \Gamma \leq 1 \qquad u(0) = \alpha \qquad u(1) = \beta$

we should like to state this problem into another form leading to a system of nonlinear equations.

Under easily verified assumptions (f continuously differentiable and $\partial_2 f(\Gamma,u) \geq 0$) problem (6.1) possesses a unique twice continuously differentiable solution. We shall study briefly two ways of numerically solving this problem.

Discretization Method

Let $\Gamma_j = jh$ with $h = \frac{1}{(n+1)}$ for $j = 0,\ldots,n+1$. At each point $\Gamma_j, j = 1,\ldots,n$ we can approximate $u''(\Gamma_j)$ by the second central difference quotient:

$$(6.2)\quad u''(\Gamma_j) = \frac{1}{h^2}[u(\Gamma_{j+1}) - 2u(\Gamma_j) + u(\Gamma_{j-1})]\ j = 1,\ldots,n$$

Introducing (6.2) in (6.1) we find that u satisfies

$$(6.3)\quad \frac{1}{h^2}[u(\Gamma_{j+1}) - 2u(\Gamma_j) + u(\Gamma_{j-1})] = f(\Gamma_j,u(\Gamma_j)) + 0(\Gamma_j,h) \quad j = 1,\ldots,n$$

where $0(\Gamma_j,h)$ is the error function due to the approximation of the second derivative. Since that error is of the order of h^2 we can assume that $0(\Gamma_j,h)$ is of the order of h^2. If x_i is the approximation to $u(\Gamma_i)$, (6.3) gives us a system of nonlinear equations

$$(6.4)\quad \begin{cases} -x_{j+1} + 2x_j - x_{j-1} + h^2 f(\Gamma_j,x_j) = 0\ ,\ j = 1,\ldots,n \\ x_0 = \alpha \quad x_{n+1} = \beta \end{cases}$$

If we define the nxn matrix A and the mapping $\phi: \mathbb{R}^n \to \mathbb{R}^n$

$$(6.5) \quad A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & 0 \\ & \ddots & \ddots & \ddots & \\ & 0 & \ddots & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \qquad \phi(x) = h^2 \begin{bmatrix} f(\Gamma_1, x_1) - \frac{\alpha}{h^2} \\ f(\Gamma_2, x_2) \\ \vdots \\ \vdots \\ f(\Gamma_n, x_n) - \frac{\beta}{h^2} \end{bmatrix}$$

then (6.4) becomes

$$(6.6) \quad F(x) = Ax + \phi(x) = 0$$

Hence the two point boundary value problem can be transformed into the problem of finding the solution of an almost linear mapping. We shall now study another way to derive a system of nonlinear equations from a two point boundary value problem.

<u>Hammerstein integral equation</u>

Let us consider the integral equation

$$(6.7) \quad u(s) = \psi(s) = \int_0^1 H(s,\Gamma)\ f(\Gamma, u(\Gamma))\ d\Gamma$$

It is not difficult to verify that if

$$(6.8) \quad \psi(s) = \alpha + (\beta - \alpha)s$$

$$(6.9) \quad H(s,\Gamma) = \begin{cases} -s(1-\Gamma) & s \leq \Gamma \\ -\Gamma(1-s) & s \geq \Gamma \end{cases}$$

then the solution of (6.7) satisfies

(6.10) $u''(s) = f(s,u) \quad u(0) = \alpha \quad u(1) = \beta$

In general the discrete analog of (6.7) is

$$u(\Gamma_i) = \Psi(\Gamma_i) + \int_0^1 H(\Gamma_i,\Gamma) f(\Gamma,u(\Gamma))d\Gamma$$

$$\triangleq \psi(\Gamma_i) + \sum_{j=1}^{n} \gamma_j H(\Gamma_i,\Gamma_j) f(\Gamma_j,u(\Gamma_j))$$

$$\xi_i = u(\Gamma_i) = \psi(\Gamma_i) + \sum_{j=1}^{n} \gamma_j \, h_{ij} \, f(\Gamma_j,\xi_j) \quad 1 \le i \le n$$

or equivalently

(6.11) $f_i(\xi_1,\ldots,\xi_n) = \xi_i - \sum_{j=1}^{n} \gamma_j \, h_{ij} f(\Gamma_j,\xi_j) - \psi(\Gamma_i) = 0$

In matrix form we get

(6.12) $F(x) = x - H\,\phi(x) - b$

where

$$b = \begin{bmatrix} \psi(\Gamma_1) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \psi(\Gamma_n) \end{bmatrix}, \qquad \phi(x) = \begin{bmatrix} \gamma_1 f(\Gamma_1,\xi_1) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \gamma_n f(\Gamma_n,\xi_n) \end{bmatrix}$$

Note that if H is nonsingular then $F(x) = 0$ if and only if

(6.13) $H^{-1}x - \phi(x) - H^{-1}b = 0$

If we choose the Γ_j to be $\Gamma_j = jh \quad h = \frac{1}{n+1} \quad j = 1,\ldots,n$
and the quadrature coefficients to be $\gamma_j = h \quad j = 1,\ldots,n$
then (6.11) becomes

$$(6.14)\quad \xi_i - \alpha - i(\beta-\alpha)h + h^2 \sum_{j=1}^{i} j(1-ih)f(jh,\xi_j) + h^2 \sum_{j=i+1}^{n} i(1-jh)f(jh,\xi_j) = 0 \qquad i = 1,\dots,n$$

which is a full system of nonlinear equations. It turns out that in this case (6.6) and (6.13) describe the same system of equations.

<u>Applications of these methods to two examples</u>

Let us consider first the following two point boundary value problem

$$u''(\Gamma) = e^{u(\Gamma)} \quad 0 \le \Gamma \le 1 \quad u(0) = u(1) = 0$$

If we apply the discretization method we get:

$$\begin{cases} x_{j+1} - 2x_j + x_{j-1} - h^2 e^{x_j} = 0 \\ x_0 = 0 \qquad x_{n+1} = 0 \qquad\qquad j = 1,\dots,n \end{cases}$$

while if we apply the Hammerstein integral equation method we get

$$\begin{cases} x_i + h^2 \sum_{j=1}^{i} j(1-ih)e^{x_j} + h^2 \sum_{j=i+1}^{n} i(1-jh)e^{x_j} = 0 \\ x_0 = 0 \qquad x_{n+1} = 0 \qquad\qquad i = 1,\dots,n \end{cases}$$

Since $e^{u(x)}$ is always positive the solution $u(x)$ will be convex. Hence we take as an initial guess

$$x_i^0 = \Gamma_i(\Gamma_i - 1) \quad i = 0,\dots, n+1 \quad \Gamma_i = ih \quad h = \frac{1}{(n+1)}$$

Let us consider now a second two point boundary value problem:

$$u''(\Gamma) = \frac{1}{2}[u(\Gamma) + \Gamma + 1]^3 \quad 0 \leq \Gamma \leq 1 \quad u(0) = u(1) = 0$$

If we apply the discretization method we get:

$$\begin{cases} x_{j+1} - 2x_j + x_{j-1} - \frac{h^2}{2}(x_j + \Gamma_j + 1)^3 = 0 \\ x_0 = 0 \qquad x_{n+1} = 0 \qquad\qquad j = 1,\dots,n \end{cases}$$

while with the Hammerstein integral equation method we get

$$\begin{cases} x_i + \frac{h^2}{2} \sum_{j=1}^{i} j(1-ih)(x_j+\Gamma_j+1)^3 + \frac{h^2}{2} \sum_{j=i+1}^{n} i(1-jh)(x_j+\Gamma_j+1)^3 = 0 \\ x_0 = 0 \qquad x_{n+1} = 0 \qquad\qquad i = 1,\dots,n \end{cases}$$

As in the previous example we take as an initial guess

$$x_i^0 = \Gamma_i(\Gamma_i - 1) \qquad i = 0,\dots,n+1 \quad \Gamma_i = ih \quad h = \frac{1}{(n+1)}$$

although the same remark does not apply.

<u>Remarks</u>

It is important to note that in the discretization method the Jacobian matrix is always tridiagonal so that the algorithms we are interested in will be less efficient than specific algorithms while the integral equation method leads to a full Jacobian matrix.

These four examples are very interesting in the sense that we can take n as big as we want so that considerations upon storage and CPU time will have more significance than for n small.

7. NUMERICAL RESULTS

In this section we shall present the numerical result for the four methods studied throughout this paper. Do not that the results are for particular implementations of thes methods and that different implementations can produce drastically different results. The four subroutines under consideration are:

ZSYSTM: a subroutine of the IMSL library implementin Brown's method in its original version.

MVOBM: a subroutine implementing the modified version of Brown's method as presented in Sections 3.2 and 5.2. A listing of this subroutine is given in Appendix 1.

NS01A: A subroutine implementing Powell's hybrid method. A listing of this subroutine can be found in [16].

BRENT: A subroutine implementing Brent's method as presented in Sections 2 and 5-1. A listing of this subroutine is given in Appendix 2.

Given a starting point and a precision EPS we compare the subroutines on the number of component function evaluat required to produce an iterate x^* such that $||F(x^*)||_2 \le$ EP However, these subroutines have different stopping criteria and other parameters which influence the behavior of the al rithm so we now describe these features. We restrict ourselves to the stopping criteria which lead to a successful termination.

ZSYSTM This subroutine stops if:

$$FNORM = \max\{|f_k(y_k)| : 1 \le k \le n\}$$

where y_k is the kth subiterate, does not exceed EPS, or when:

$$||x_{k+1} - x_k||_\infty \le 10^{-NSIG}\,||x_k||_\infty$$

We made sure that the second stopping criterion was never satisfied by choosing NSIG = 70.

A point of interest is that ZSYSTM has a post-solution testing in which if for some k the kth component of the final iterate is near an integer, then this component is set to that particular integer if this action decreases the kth residual.

MVOBM This subroutine stops if FNORM $\le$ EPS or if

$$||x_{k+1} - x_k||_\infty \le TOL\,||x_k||_\infty$$

The second stopping criterion was disabled by choosing TOL = 10^{-70}.

NS01A This subroutine stops if $||F(x^*)||_2 \le$ EPS. However, the calling sequence has two other parameters: DMAX and DSTEP. The choice of DMAX does not seem to be very critical (a good choice is an estimate of the initial guess to the solution) but different choices of DSTEP have a striking effect on the behavior of the algorithm. Therefore, for each function we have listed the results for the best value of DSTEP.

BRENT This subroutine stops if FNORM $\leq$ EPS or if $||x_{k+1} - x_k||_\infty \leq TOL\ ||x_k||_\infty$

The second stopping criterion was disabled by choosing TOL = 10^{-70}.

7.1 Rosenbrock's function

We tested each method on Rosenbrock's function with precisions of 10^{-5} and 10^{-10}. For each precision three different guesses were used:

$$x_0 = (-1.2,1.0)^T \quad x_0 = 10(-1.2,1.0)^T \quad x_0 = 100(-1.2,1.0)^T$$

The results are listed in Figures 1 and 2.

We note that MVOBM, ZSYSTM and BRENT performed better than NS01A (DSTEP = 10^{-6}) whatever the starting point and precision. This is due to the particular structure of the problem: the Jacobian matrix is lower triangular. MVOBM and BRENT gave exactly the same results for all tests and this can be explained by the same remark.

Also note that the post-solution testing in ZSYSTM took place in some of the cases.

<u>Ranking:</u> 1^{st} MVOBM and BRENT

3^{rd} ZSYSTM

4^{th} NS01A (DSTEP = 10^{-6})

Figure 1 - Rosenbrock's Function

(a) Precision = 10^{-5} $x_0 = (-1.2,1.0)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	10	2	0.0
ZSYSTM	20	3	0.0
BRENT	10	2	0.0
NSO1A	54	24	0.0

(b) Precision = 10^{-5} $x_0 = 10(-1.2,1.0)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	10	2	0.0
ZSYSTM	13	2	0.0
BRENT	10	2	0.0
NSO1A	88	41	0.0

(c) Precision = 10^{-5} $x_0 = 100(-1.2,1.0)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	10	2	0.0
ZSYSTM	13	2	0.0
BRENT	10	2	0.0
NSO1A	486	240	$7.7\ 10^{-13}$

Figure 2 - Rosenbrock's Function

(a) Precision = 10^{-10} $x_0 = (-1.2,1.0)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	15	3	0.0
ZSYSTM	20	3	0.0
BRENT	15	3	0.0
NSO1A	54	24	0.0

(b) Precision = 10^{-10} $x_0 = 10(-1.2,1.0)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	15	3	0.0
ZSYSTM	13	2	0.0
BRENT	15	3	0.0
NSO1A	88	41	0.0

(c) Precision = 10^{-10} $x_0 = 10(-1.2,1.0)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	15	3	0.0
ZSYSTM	25	4	0.0
BRENT	15	3	0.0
NSO1A	426	240	7.7 10^{-11}

7.2 Powell's singular function

We tested each method on Powell's singular function with precisions of 10^{-5} and 10^{-10}. For each precision three different starting points were used:

$$x_0 = (3,-1,0,1)^T \quad x_0 = 10(3,-1,0,1)^T \quad x_0 = 100(3,-1,0,1)^T$$

The results are listed in Figures 3 and 4.

Since the Jacobian matrix at the solution is singular, the convergence is not quadratic; instead, it is only linear. This explains why, in each case, the numbers of function evaluations do not differ too much.

Finally note that ZSYSTM seemed to have more trouble than the other subroutines in going from a precision of 10^{-5} to that of 10^{-10}.

Ranking: 1^{st} NS01A (DSTEP = 10^{-6})
2^{nd} BRENT
3^{rd} MVOBM
4^{th} ZSYSTM

Figure 3 - Powell's singular function

(a) Precision = 10^{-5} $x_0 = (3,-1,0,1)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	196	14	1.2 10^{-6}
ZSYSTM	203	14	0.0
BRENT	182	13	1.9 10^{-6}
NSO1A	104	21	4.0 10^{-6}

(b) Precision = 10^{-5} $x_0 = 10(3,-1,0,1)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	252	18	7.0 10^{-7}
ZSYSTM	231	16	0.0
BRENT	238	17	1.0 10^{-6}
NSO1A	288	67	4.0 10^{-6}

(c) Precision = 10^{-5} $x_0 = 100(3,-1,0,1)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	294	21	1.4 10^{-6}
ZSYSTM	287	20	0.0
BRENT	280	20	2.0 10^{-6}
NSO1A	264	61	8.2 10^{-6}

Figure 4 - Powell's singular function

(a) Precision = 10^{-10} $x_0 = (3,-1,0,1)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	322	23	1.0 10^{-11}
ZSYSTM	438	31	0.0
BRENT	308	22	1.5 10^{-11}
NSO1A	272	63	5.2 10^{-11}

(b) Precision = 10^{-10} $x_0 = 10(3,-1,0,1)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	364	26	2.0 10^{-11}
ZSYSTM	508	36	0.0
BRENT	364	26	7.3 10^{-12}
NSO1A	336	80	4.0 10^{-11}

(c) Precision = 10^{-10} $x_C = 100(3,-1,0,1)^T$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	420	30	1.0 10^{-11}
ZSYSTM	564	40	0.0
BRENT	406	29	1.3 10^{-11}
NSO1A	352	83	5.6 10^{-11}

7.3 Chebyquad functions

We tested each method on the Chebyquad functions for $n = 3,5,7,9$, and for precisions of 10^{-5} and 10^{-10}. For each precision three different starting points were used:

$$X_0 = u, \quad X_0 = 10u \quad \text{and} \quad X_0 = 100u$$

where: $u = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$.

The results are listed in figures 5 thru 12.

It is interesting to note that the algorithms did not always converge to the same solution. This is due to the fact that each function has n! solutions. It turns out that convergence is easy when we start with $x_0 = u$, but it becomes more difficult as we get farther away from the solution since the algorithms seem to hesitate between solutions.

Overall, BRENT outperformed the other subroutines although when $x_0 = u$, MVOBM was comparable. On the other hand, ZSYSTM and NSO1A (with any choice of DSTEP) seemed to have a great deal of trouble converging.

In Figures 5 thru 12:

(*) means that the subroutine stopped because it met twice a singular Jacobian (ZSYSTM)

(**) means that n+4 consecutive iterations failed to improve the residuals (NSO1A)

(***) means that a nearby stationary point of F(x) was predicted (NSO1A).

- The solutions of the Chebyquad functions for n = 3,5,7 and 9 are (recall that it is relative to a permutation of the components):

n = 3 $x_1^* = 0.146446609407$ $x_2^* = 0.5$
$x_3^* = 0.853553390593$

n = 5 $x_1^* = 0.0837512564995$ $x_2^* = 0.312729295223$
$x_3^* = 0.5$ $x_4^* = 0.687270704777$
$x_5^* = 0.916248743500$

n = 7 $x_1^* = 0.0580691496210$ $x_2^* = 0.235171612358$
$x_3^* = 0.338044094740$ $x_4^* = 0.5$
$x_5^* = 0.661955905260$ $x_6^* = 0.764828387642$
$x_7^* = 0.941930850379$

n = 9 $x_1^* = 0.0442053461358$ $x_2^* = 0.199490672310$
$x_3^* = 0.235619108471$ $x_4^* = 0.416046907893$
$x_5^* = 0.5$ $x_6^* = 0.583953092107$
$x_7^* = 0.764380891529$ $x_8^* = 0.800509327690$
$x_9^* = 0.955794653864$

Figure 5 - Chebyquad n = 3

(a) Precision = 10^{-5} $x_0 = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	36	4	1.5 10^{-12}
ZSYSTM	20(*)		2
BRENT	36	4	1.3 10^{-12}
NSO1A	78(*)		1.3 10^{-3}

(b) Precision = 10^{-5} $x_0 = 10(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	90	10	2.6 10^{-16}
ZSYSTM	119	13	1.2 10^{-15}
BRENT	63	7	3.2 10^{-15}
NSO1A	219	69	6.5 10^{-6}

(c) Precision = 10^{-5} $x_0 = 100(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	126	14	8.2 10^{-17}
ZSYSTM	92(*)		1.4 10^{-3}
BRENT	90	10	5.9 10^{-13}
NSO1A	57(**)		5.9 10^{-6}

Figure 6 - Chebyquad n = 3

(a) Precision = 10^{-10} $x_0 = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	45	5	5.9 10^{-17}
ZSYSTM	20 (*)		2.0
BRENT	45	5	2.5 10^{-16}
NSO1A	78 (*)		1.3 10^{-3}

(b) Precision = 10^{-10} $x_0 = 10(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	90	10	2.6 10^{-16}
ZSYSTM	137 (*)		7.1
BRENT	72	8	3.5 10^{-16}
NSO1A	438	142	9.3 10^{-11}

(c) Precision = 10^{-10} $x_0 = 100(\frac{n}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	135	15	5.6 10^{-17}
ZSYSTM	92 (*)		1.4 10^{-3}
BRENT	99	11	6.2 10^{-17}
NSO1A	57 (**)		5.9 10^{-6}

Figure 7 - Chebyquad n = 5

(a) Precision = 10^{-5} $x_0 = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	100	5	5.4 10^{-11}
ZSYSTM	37(*)		2.7 10^{-1}
BRENT	80	4	4.0 10^{-15}
NSO1A	30(***)		5.9 10^{6}

(b) Precision = 10^{-5} $x_0 = 10(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	980	49	3.0 10^{-12}
ZSYSTM	157(*)		2.1 10^{14}
BRENT	200	10	1.3 10^{-16}
NSO1A	1085(***)		1.7 10^{-4}

(c) Precision = 10^{-5} $x_0 = 100(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	1640	82	3.2 10^{-16}
ZSYSTM	317(*)		4.2 10^{15}
BRENT	300	15	6.7 10^{-15}
NSO1A	DIV		DIV

Figure 8 - Chebyquad n = 5

(a) Precision = 10^{-10} $x_0 = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	120	6	1.4 10^{-16}
ZSYSTM	37(*)		2.7 10^{-1}
BRENT	100	5	1.4 10^{-16}
NS01A	30(***)		5.9 10^{6}

(b) Precision = 10^{-10} $x_0 = 10(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	1000	50	3.1 10^{-16}
ZSYSTM	157(*)		2.1 10^{14}
BRENT	220	11	1.9 10^{-16}
NS01A	1025(**)		1.7 10^{-4}

(c) Precision = 10^{-10} $x_0 = 100(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	1660	83	3.4 10^{-16}
ZSYSTM	317(*)		4.0 10^{15}
BRENT	320	16	1.5 10^{-16}
NS01A	DIV		DIV

Figure 9 - Chebyquad n = 7

(a) Precision = 10^{-5} $x_0 = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	140	4	5.4 10^{-11}
ZSYSTM	58(*)		3.5 10^{-1}
BRENT	140	4	2.2 10^{-13}
NSO1A	329(**)		1.6 10^{-2}

(b) Precision = 10^{-5} $x_0 = 10(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
ZSYSTM	DIV		DIV
BRENT	1435	41	4.0 10^{-11}
NSO1A	1925(max)		3.0 10^{-1}

(c) Precision = 10^{-5} $x_0 = 100(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
ZSYSTM	DIV		DIV
BRENT	1120	32	6.4 10^{-11}
NSO1A	DIV		DIV

Figure 10 - Chebyquad n = 7

(a) Precision = 10^{-10} $x_0 = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	175	5	$3.0 \quad 10^{-16}$
ZSYSTM	58(*)		$3.5 \quad 10^{-1}$
BRENT	175	5	$2.6 \quad 10^{-16}$
NSO1A	329(**)		$1.6 \quad 10^{-2}$

(b) Precision = 10^{-10} $x_0 = 10(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
ZSYSTM	DIV		DIV
BRENT	1470	42	$2.5 \quad 10^{-16}$
NSO1A	1925(max)		$3.0 \quad 10^{-1}$

(c) Precision = 10^{-10} $x_0 = 100(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
ZSYSTM	DIV		DIV
BRENT	1155	33	$2.6 \quad 10^{-16}$
NSO1A	DIV		DIV

Figure 11 - Chebyquad n = 9

(a) Precision = 10^{-5} $x_0 = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	324	6	2.9 10^{-15}
ZSYSTM	83(*)		6.4 10^{-2}
BRENT	270	5	1.8 10^{-13}
NSO1A	522(**)		2.0 10^{-2}

(b) Precision = 10^{-5} $x_0 = 10(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
ZSYSTM	DIV		DIV
BRENT	DIV		DIV
NSO1A	DIV		DIV

(c) Precision = 10^{-5} $x_0 = 100(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
ZSYSTM	DIV		DIV
BRENT	DIV		DIV
NSO1A	DIV		DIV

Figure 12 - Chebyquad n = 9

(a) Precision = 10^{-10} $x_0 = (\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	378	7	3.9 10^{-16}
ZSYSTM	83(*)		6.4 10^{-2}
BRENT	324	6	4.2 10^{-16}
NSO1A	522(**)		2.0 10^{-2}

(b) Precision = 10^{-10} $x_0 = 10(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
ZSYSTM	DIV		DIV
BRENT	DIV		DIV
NSO1A	DIV		DIV

(c) Precision = 10^{-10} $x_0 = 100(\frac{1}{n+1}, \ldots, \frac{n}{n+1})$

	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
ZSYSTM	DIV		DIV
BRENT	DIV		DIV
NSO1A	DIV		DIV

7.4 Differential Equation I. Discretization Method.

We tested each method on the function obtained by the discretization of the two point boundary value problem.

$$u''(t) = e^{u(t)} \quad , 0 \leq t \leq 1$$
$$u(0) = u(1) = 0$$

As mentioned in Section 6.2, the standard discretization of this problem leads to a system of n equations in n unknowns. We solved this sytem for n = 10, 50 and 100. Moreover, since the error in this discretization is $0(h^2)$ where $h = 1/(n+1)$; our stopping criterion was h^2.

We have listed the results in Figure 13. In addition to the usual information, we have also listed the CPU time and storage required by these methods. These two items, more than anything, reflect the efficiency of the methods.

The results in Figure 13 clearly indicate that the best method is MVOBM and the worst is ZSYSTM. NSO1A is better than BRENT in terms of CPU time but worse in terms of storage. However note that the CPU time of BRENT is due to the fact that it takes one iteration more than is probably necessary (see the results of MVOBM) and therefore gives a smaller final residual than NSO1A.

Ranking: 1st MVOBM
2nd NSO1A (DSTEP = 10^{-2}) and BRENT
4th ZSYSTM

Figure 13 - Differential Equation 1 - Discretization Method

(a) n = 10, precision = $8,10^{-3}$

	STORAGE	CPUTIME (sec)	FEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	32K	.06	130	2	$3.2\ 10^{-9}$
ZSYSTM	33K	.13	139	2	$1.3\ 10^{-4}$
BRENT	34K	.10	130	2	$1.8\ 10^{-6}$
NS01A	46K	.14	170	6	$3.3\ 10^{-5}$

(b) n = 50, precision = 4.10^{-4}

	STORAGE	CPUTIME (sec)	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	40K	.93	2,650	2	$3.2\ 10^{-10}$
ZSYSTM	48K	17.14	2,699	2	$3.1\ 10^{-5}$
BRENT	54K	3.38	3,975	3	$7.3\ 10^{-13}$
NS01A	106K	2.05	2,900	7	$3.7\ 10^{-6}$

(c) n = 100, precision = 1.10^{-4}

	STORAGE	CPUTIME (sec)	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	56K	5.52	10,300	2	$1.3\ 10^{-10}$
ZSYSTM	78K	No convergence after 57.23 seconds (max)			
BRENT	114K	26.11	15,450	3	$1.0\ 10^{-15}$
NS01A	286K	11.87	10,900	8	$5.3\ 10^{-7}$

7.5 Differential Equation II. Discretization Method.

We tested each method on the function obtained by the discretization of the two point boundary value problem

$$u''(t) = \frac{1}{2}(u(t) + t + 1)^3 \qquad 0 \le t \le 1$$
$$u(0) = u(1) = 0$$

and solved the resulting system of nonlinear equations for $n = 10, 50, 100$. The results are listed in Figure 14, and are entirely analogous to those in Section 7.4.

Ranking: 1^{st} MVOBM
2^{nd} NSO1A (DSTEP = 10^{-2}) and BRENT
4^{th} ZSYSTM

Figure 14 - Differential Equation 2 - Distretization Method

(a) n = 10, precision 8.10^{-3}

	STORAGE	CPUTIME (sec)	FEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	32K	.06	130	2	$3.8\ 10^{-7}$
ZSYSTM	33K	.13	139	2	$1.0\ 10^{-4}$
BRENT	34K	.10	130	2	$1.7\ 10^{-6}$
NSO1A	46K	.12	150	4	$5.0\ 10^{-3}$

(b) n = 50, precision 4.10^{-4}

	STORAGE	CPUTIME (sec)	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	40K	.95	2,650	2	$3.8\ 10^{-8}$
ZSYSTM	46K	17.71	2,699	2	$2.0\ 10^{-5}$
BRENT	54K	3.90	3,975	3	$5.6\ 10^{-15}$
NSO1A	106K	2.04	2,850	6	$2.3\ 10^{-6}$

(c) n = 100, precision 1.10^{-4}

	STORAGE	CPUTIME (sec)	FEVALS	NITER	$\|\|F(x^*)\|\|_2$
MVOBM	56K	8.30	15,450	3	$1.4\ 10^{-13}$
ZSYSTM	78K	No convergence after 57.11 seconds (max)			
BRENT	114K	25.57	15,450	3	$2.3\ 10^{-15}$
NSO1A	284K	10.94	10,700	6	$9.5\ 10^{-5}$

7.6 Differential Equation I. Integral Equation Method.

We tested the routines MVOBM, BRENT, and NS01A on the function obtained by transforming the two-point boundary value problem (7.4) into a Hammerstein integral equation.

As mentioned in Section 6.2, a discretization of the resulting integral equation leads to a system of n equations in n unknowns. We solved this system for n = 10, 50, and as in Section 6.2, we also used 100 x_0 as an initial guess.

The results are listed in Figures 15 and 16, and are somewhat different from those in Section 7.4. Note that in this case the function evaluations are more costly and reflect accurately the efficiency of the methods as measured by the CPU time. Also note that now MVOBM and BRENT require the same number of function evaluations and both require three iterations to converge. On the other hand, the convergence characteristics of NS01A improve.

Ranking: 1^{st} NS01A (DSTEP = 10^{-2})
2^{nd} MVOBM and BRENT

Figure 15 - Differential Equation 1 - Hammerstein Method N = 10

(a) Precision 8.10^{-3} starting point: x_0 as in Section 6.2

	FCNEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	130	2	10^{-8}
BRENT	130	2	8.9 10^{-9}
NSO1A	120	1	2.5 10^{-3}

(b) Precision 8.10^{-3} starting point: 100 x_0

	FCNEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	195	3	5.5 10^{-10}
BRENT	195	3	4.8 10^{-10}
NSO1A	140	3	1.1 10^{-4}

Figure 16 - Differential Equation 1. Hammerstein Method N = 50

(a) Precision 4.10^{-4} starting point: x_0 as in Section 6.2

	STORAGE	CPUTIME (sec)	FCNEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	40K	6.80	3,975	3	$1.4\ 10^{-15}$
BRENT	54K	8.65	3,975	3	$1.2\ 10^{-15}$
NSO1A	106K	5.01	2,650	2	$1.7\ 10^{-5}$

(b) Precision 4.10^{-4} starting point: $100x_0$

	STORAGE	CPUTIME (sec)	FCNEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	40K	6.80	3,975	3	$4.0\ 10^{-10}$
BRENT	54K	8.68	3,975	3	$3.5\ 10^{-10}$
NSO1A	106K	5.13	2,700	3	$2.3\ 10^{-4}$

7.7 Differential Equation II. Integral Equation Method.

We tested the routines MVOBM, BRENT and NSO1A on the function obtained by transforming the two-point boundary value problem (7.5) into a Hammerstein integral equation. We then proceeded as in Section 7.6.

The results are listed in Figures 17 and 18. As in Section 7.6, NSO1A is the best routine, but note that in this case the difference is more pronounced since both MVOBM and BRENT diverged for the farthest initial guess.

Ranking: 1st NSO1A (DSTEP = 10^{-2})
2nd MVOBM and BRENT.

Figure 17 - Differential Equation 2. Hammerstein Method. N=10

(a) Precision 8.10^{-3} Starting point: x_0 as in Section 6.2

	FCNEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	130	2	3.5 10^{-8}
BRENT	130	2	6.1 10^{-8}
NSO1A	130	2	1.3 10^{-4}

(b) Precision 8.10^{-3} Starting point: $100x_0$

	FCNEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	DIV		DIV
BRENT	DIV		DIV
NSO1A	430		8.7 10^{-4}

Figure 18 - Differential Equation 2. Hammerstein Method. N=50

(a) Precision 4.10^{-4} Starting point: x_0 as in Section 6.2

	STORAGE	CPUTIME (sec)	FCNEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	40K	3.64	3,975	3	$7.2\ 10^{-15}$
BRENT	54K	5.83	3,975	3	$5.9\ 10^{-15}$
NS01A	106K	4.62	2,700	3	$6.8\ 10^{-7}$

(b) Precision 4.10^{-4} Starting point: $100x_0$

	STORAGE	CPUTIME (sec)	FCNEVALS	NBITER	$\|\|F(x^*)\|\|_2$
MVOBM	40K	DIV			DIV
BRENT	54K	DIV			DIV
NS01A	106K	11.71	6,850		$3.7\ 10^{-4}$

8. CONCLUSION OF THE COMPARISON

When we look carefully at all the results listed in the previous pages two facts are clear:

1. The modified version of Brown's method is a real improvement of the original version. The CPU time and the storage requirement are decreased by a considerable amount and expected consequence of this fact is that the convergence properties are improved.

2. In all cases Brent's method never did worse than the second place and was often the best method.

Let us make a short review of the four methods and state their advantages and disadvantages:

Brown's original method:

It is the oldest and is now beaten by the recent methods, the modified version and Brent's method.

Powell's hybrid method

This method did less well than expected. It failed a great number of times because it could not decrease the residuals. Moreover, only in rare cases was the convergence faster than the other methods (in terms of function evaluations). However, it is a good choice when the function is smooth and when nothing is known about the starting point.

Brent's method

In the average, it is the best method. The convergence is very fast and the region of convergence is often big enough. An advantage of the method seems to be that its behavior is good for non smooth functions. When the system to solve seems ill-conditioned we recommend this method very strongly. Its disadvantages are obvious: number of arithmetic operations and storage locations. It is the reason why we tried in 5 to show a way of implementing more efficiently this method.

The modified version of Brown's method

It is a good method especially if the system is large, the function is smooth, and a good guess is available. However, in the average case it is beaten by Brent's method.

Our classification of the methods will be:

1. Brent's method
2. Brown's modified and Powell's hybrid method
3. Brown's initial method

›PENDICES

In these appendices we give the listings of Brown's ›dified method (MVOBM), Brent's method (BRENT) and the ınctions described in Section 6.

Appendix 1

```
C.MVCBM.............................................................
C
C    FUNCTION                       - DETERMINATION OF A ROOT OF A SYSTEM OF
C                                     N SIMULTANEOUS NONLINEAR EQUATIONS IN
C                                     N UNKNOWNS, F(X)=0, IN VECTOR FORM,
C                                     BY BROWN'S METHOD.
C    USAGE                          -CALL MVCBM(N,F,X,EPS,TOL,ITMAX,W,Y,Z,
C                                     MPERM,U,IER)
C    PARAMETERS     F               - F IS THE NAME OF THE DOUBLE PRECISION
C                                     FUNCTION WHICH DEFINES THE SYSTEM OF
C                                     EQUATIONS.THE USER SPECIFIES F BY
C                                     WRITING A FUNCTION SUBPROGRAM F(X,K)
C                                     WHICH COMPUTES THE K-TH COMPONENT OF
C                                     F EVALUATED AT X. (INPUT)
C                   EPS             - FIRST STOPPING CRITERION.THE MAXIMUM
C                                     ABSOLUTE VALUE OF F(X,K) IS LESS THAN
C                                     OR EQUAL TO EPS,FOR K=1,...,N.(INPUT)
C                   TOL             - SECOND STOPPING CRITERION.THE RELATIVE
C                                     ERROR(IN THE MAX-NORM) BETWEEN TWO
C                                     SUCCESSIVE ITERATES IS LESS THAN TOL.
C                   N               - THE NUMBER OF EQUATIONS (= NUMBER OF
C                                     UNKNOWNS).(INPUT)
C                   X               - THE VECTOR X OF LENGTH N,AS INPUT,IS
C                                     THE INITIAL GUESS FOR THE ROOT.AS
C                                     OUTPUT,IT IS THE COMPUTED SOLUTION.
C                   ITMAX           - ON INPUT = THE MAXIMUM ALLOWABLE
C                                     NUMBER OF ITERATIONS AND ON OUTPUT =
C                                     THE NUMBER OF ITERATIONS USED IN
C                                     FINDING THE COMPUTED SOLUTION.
C                   U               - A WORKING ARRAY OF SIZE N**2/4.
C                   W,Y,Z           - WORKING ARRAYS OF SIZE N.
C                   MPERM           - AN INTEGER WORKING ARRAY OF SIZE N.
C                   IER             - RETURN PARAMETER
C                                       IER=1 FIRST STOPPING CRITERION
C                                       IER=2 NUMBER OF ITERATIONS GREATER
C                                             THAN ITMAX.
C                                       IER=3 SECOND STOPPING CRITERION
C                                       IER=4 JACOBIAN MATRIX IS SINGULAR.
C                                       IER=5 THE NORM OF X IS GREATER THAN
C                                             HUNDRED TIMES ITS INITIAL
C                                             VALUE:DIVERGENCE.
C    PRECISION                      - DOUBLE
C    LANGUAGE                       - FORTRAN
C...................................................................
C
      SUBROUTINE MVOBM(N,F,X,EPS,TOL,ITMAX,W,Y,Z,MPERM,U,IER)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(1),W(1),Y(1),Z(1),MPERM(1),U(1)
      DATA ZERO,ONE,XMIN,HEPS /0.D0,1.D0,1.D-30,1.D-8/
C*******************************************************************
C COMPUTE THE NORM OF X.
C ITER COUNTS THE NUMBER OF ITERATIONS.
C KONFT IS USED TO CHECK IF THE JACOBIAN IS VERY SINGULAR
C MPERM CONTAINS THE PERMUTATION VECTOR OF GAUSSIAN ELIMINATION WITH
```

```
C PARTIAL PIVOTING. IT IS INITIALIZED BY MPERM(I)=I .                   *
C Y IS THE SUBITERATE AND IS INITIALIZED WITH X.                        *
C***********************************************************************
      IER=0
      ITER=0
      XNORM=ZERO
      DO 10 I=1,N
               AUX=DABS(X(I))
               IF (AUX.GT.XNORM) XNORM=AUX
               MPERM(I)=I
               Y(I)=X(I)
10             CONTINUE
      XMIN=100.D0*DMAX1(XNORM,ONE)
C***********************************************************************
C ENTER THE PRINCIPAL ITERATION.                                        *
C***********************************************************************
20    CONTINUE
      ITER=ITER+1
      FNORM=ZERO
      KCRET=0
C***********************************************************************
C COMPUTE THE STEP H FOR THE DIVIDED DIFFERENCE FORMULA TO APPROXIMATE*
C   THE DERIVATIVE.                                                     *
C***********************************************************************
      H=REPS*DMAX1(XNORM,XMIN)
C***********************************************************************
C ENTER A SUBITERATION.                                                 *
C***********************************************************************
      DO 110 K=1,N
               NMK=N-K
               NMKP1=NMK+1
               KP1=K+1
               KM1=K-1
               FONCT=F(K,Y)
               AUX=DABS(FONCT)
               IF (AUX.GT.FNORM) FNORM=AUX
C***********************************************************************
C COMPUTE THE KTH ROW OF THE JACOBIAN MATRIX.                           *
C***********************************************************************
               DO 45 J=1,NMKP1
                         DO 30 I=1,N
                                   Z(I)=Y(I)
30                                 CONTINUE
                         IF (K.EQ.1) GOTO 40
                         IND=J
                         DO 35 I=1,KM1
                                   INDEX=MPERM(I)
                                   Z(INDEX)=Z(INDEX)+U(IND)
                                   IND=IND+NMKP1
35                                 CONTINUE
40                       CONTINUE
                         IND=J+KM1
                         INDEX=MPERM(IND)
                         Z(INDEX)=Z(INDEX)+H
                         W(IND)=F(K,Z)-FONCT
45                       CONTINUE
```

```
C*********************************************************************
C   COMPUTE THE MAX OF W FROM K TO N AND PERMUTE THE ELEMENTS SO THAT
C THE MAX IS THE KTH ELEMENT OF W.THIS CORRESPONDS TO THE PARTIAL
C PIVOTING OF GAUSSIAN ELIMINATION.
C*********************************************************************
                  JSTAR=K
                  AUX=W(K)
                  DO 50 J=K,N
                           IF (DABS(W(J)).LE.DABS(AUX)) GOTO  50
                           JSTAR=J
                           AUX=W(J)
      50                   CONTINUE
                  IF (JSTAR.EQ.K) GOTO 60
                  INDEX=MPERM(K)
                  MPERM(K)=MPERM(JSTAR)
                  MPERM(JSTAR)=INDEX
                  W(JSTAR)=W(K)
                  W(K)=AUX
C*********************************************************************
C PERMUTE THE ELEMENTS OF THE VECTOR U.
C*********************************************************************
                  IF (K.EQ.1) GOTO 60
                  INDEX=1
                  IND=JSTAR-KM1
                  DO 55 I=1,KM1
                           UHELP=U(IND)
                           U(IND)=U(INDEX)
                           U(INDEX)=UHELP
                           IND=IND+NMKP1
                           INDEX=INDEX+NMKP1
      55                   CONTINUE
C*********************************************************************
C COMPUTE THE NEW VALUE OF THE NEXT SUBITERATE.
C*********************************************************************
      60          CONTINUE
                  AUX=ZERO
                  IF (W(K).EQ.ZERO) KOMPT=KOMPT+1
                  IF (W(K).NE.ZERO)  AUX=FONCT/W(K)
                  IF (K.EQ.1) GOTO 70
                  IND=1
                  DO 65 I=1,KM1
                           INDEX=MPERM(I)
                           Y(INDEX)=Y(INDEX)-AUX*U(IND)
                           IND=IND+NMKP1
      65                   CONTINUE
      70          CONTINUE
                  INDEX=MPERM(K)
                  Y(INDEX)=Y(INDEX)-H*AUX
C*********************************************************************
C COMPUTE THE NEW VALUE OF U.
C*********************************************************************
                  IF (K.EQ.N) GOTO 110
                  AUX=W(K)
                  IF (AUX.EQ.ZERO) GOTO 80
                  DO 75 I=KP1,N
                           W(I)=W(I)/AUX
```

```
75                    CONTINUE
80              CONTINUE
                INDEX=1
                II=1
                IF (K.EQ.1) GOTO 100
                DO 95 I=1,KM1
                      UHELP=U(INDEX)
                      DO 90 J=1,NMK
                            U(II)=U(INDEX+J)-UHELP*W(K+J)
                            II=II+1
90                          CONTINUE
                      INDEX=INDEX+NMKP1
95                    CONTINUE
100             CONTINUE
                DO 105 I=KP1,N
                      U(II)=-H*W(I)
                      II=II+1
105                   CONTINUE
110             CONTINUE
C*********************************************************************
C COMPUTE THE NORM OF THE ITERATE AND THE NORM OF THE DIFFERENCE     *
C BETWEEN THE TWO LAST ITERATES.                                     *
C*********************************************************************
      XNORM=ZERO
      DIFIT=ZERO
      DO 115 I=1,N
                YI=Y(I)
                AUX=DABS(YI)
                IF (AUX.GT.XNORM) XNORM=AUX
                AUX=DABS(X(I)-YI)
                IF (AUX.GT.DIFIT) DIFIT=AUX
                X(I)=YI
                MEERM(I)=I
115             CONTINUE
C*********************************************************************
C STOPPING CRITERIA.                                                 *
C*********************************************************************
      IF (KOMPT.GE.N) IER=4
      IF (ENORM.LT.EPS) IER=1
      IF (ITER.GE.ITMAX) IER=2
      IF (DIFIT.LT.(TOL*XNORM)) IER=3
      IF (XNORM.GT.XNMI) IER=5
      IF (IER.EQ.0) GOTO 20
      ITMAX=ITER
      RETURN
      END
```

Appendix 2

```
C.BRENT.................................................................
C
C    FUNCTION                      - DETERMINATION OF A ROOT OF A SYSTEM OF
C                                     N SIMULTANEOUS NONLINEAR EQUATIONS IN
C                                     N UNKNOWNS, F(X)=0, IN VECTOR FORM,
C                                     BY BRENT'S METHOD.
C    USAGE                         - CALL BRENT(N,F,X,EPS,TOL,ITMAX,Q,Y,Z,A,
C                                     IER).
C    PARAMETERS     F              - F IS THE NAME OF THE DOUBLE PRECISION
C                                     FUNCTION WHICH DEFINES THE SYSTEM OF
C                                     EQUATIONS.THE USER SPECIFIES F BY
C                                     WRITING A FUNCTION SUBPROGRAM F(X,K)
C                                     WHICH COMPUTES THE K-TH COMPONENT OF
C                                     F EVALUATED AT X. (INPUT)
C                   EPS            - FIRST STOPPING CRITERION.THE MAXIMUM
C                                     ABSOLUTE VALUE OF F(X,K) IS LESS THAN
C                                     OR EQUAL TO EPS,FOR K=1,...,N.(INPUT)
C                   TOL            - SECOND STOPPING CRITERION.THE RELATIVE
C                                     ERROR(IN THE MAX-NORM) BETWEEN TWO
C                                     SUCCESSIVE ITERATES IS LESS THAN TOL.
C                   N              - THE NUMBER OF EQUATIONS (= NUMBER OF
C                                     UNKNOWNS).(INPUT)
C                   X              - THE VECTOR X OF LENGTH N,AS INPUT,IS
C                                     THE INITIAL GUESS FOR THE ROOT.AS
C                                     OUTPUT,IT IS THE COMPUTED SOLUTION.
C                   ITMAX          - ON INPUT = THE MAXIMUM ALLOWABLE
C                                     NUMBER OF ITERATIONS AND ON OUTPUT =
C                                     THE NUMBER OF ITERATIONS USED IN
C                                     FINDING THE COMPUTED SOLUTION.
C                   Q              - AN N*N WORKING ARRAY.
C                   Y,Z,A          - WORKING ARRAYS OF SIZE N.
C                   IER            - RETURN PARAMETER
C                                     IER=1 FIRST STOPPING CRITERION
C                                     IER=2 NUMBER OF ITERATIONS GREATER
C                                           THAN ITMAX.
C                                     IER=3 SECOND STOPPING CRITERION
C                                     IER=4 JACOBIAN MATRIX IS SINGULAR.
C                                     IER=5 THE NORM OF X IS GREATER THAN
C                                           HUNDRED TIMES ITS INITIAL
C                                           VALUE:DIVERGENCE.
C    PRECISION                     - DOUBLE
C    LANGUAGE                      - FORTRAN
C.......................................................................
C
      SUBROUTINE BRENT(N,F,X,EPS,TOL,ITMAX,Q,Y,Z,A,IER)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(N),Y(N),Z(N),A(N),Q(N,N)
      DATA ZERO,ONE,XMIN,HEPS /0.D0,1.D0,1.D-30,1.D-7/
C***********************************************************************
C    ITER COUNTS THE NUMBER OF ITERATIONS.
C    FNORM IS THE INFINITY NORM OF F(K,X).
C    KONST IS USED WHEN THE JACOBIAN MATRIX IS VERY SINGULAR.
C    XNORM IS THE INFINITY NORM OF X.
C***********************************************************************
```

```
      IER=0
      ITER=0
      XNORM=ZERO
      DO 5 I=1,N
              DO 1 J=1,N
                      Q(I,J)=ZERO
1                     CONTINUE
              Q(I,I)=ONE
              AUX=DABS(X(I))
              IF (AUX.GT.XNORM) XNORM=AUX
              Y(I)=X(I)
5             CONTINUE
      XMIN=100.D0*DMAX1(XNORM,ONE)
C*****************************************************************
C   ENTER THE PRINCIPAL ITERATION.                               *
C*****************************************************************
10    CONTINUE
      ITER=ITER+1
      KCNET=0
      FNORM=ZERO
C*****************************************************************
C   COMPUTE THE STEP H FOR THE DIVIDED DIFFERENCE FORMULA TO     *
C APPROXIMATE THE DERIVATIVE.                                    *
C*****************************************************************
      H=REPS*DMAX1(XNORM,XMIN)
      HH=ONE/H
      DO 20 I=1,N
              DO 20 J=1,N
                      Q(I,J)=H*Q(I,J)
20                    CONTINUE
C*****************************************************************
C   ENTER A SUBITERATION.                                        *
C*****************************************************************
      DO 70 K=1,N
              KP1=K+1
C*****************************************************************
C COMPUTE THE K-TH ROW OF THE JACOBIAN MATRIX.                   *
C*****************************************************************
              FONCT=F(K,Y)
              AUX=DABS(FONCT)
              IF (AUX.GT.FNORM) FNORM=AUX
              DO 30 J=K,N
                      DO 25 I=1,N
                              Z(I)=Y(I)+Q(I,J)
25                            CONTINUE
                      A(J)=F(K,Z)-FONCT
30                    CONTINUE
C*****************************************************************
C   COMPUTE THE HOUSEHOLDER TRANSFORMATION TO REDUCE THE K-TH ROW *
C   OF THE JACOBIAN MATRIX TO A MULTIPLE OF THE K-TH UNIT VECTOR. *
C*****************************************************************
              ETA=ZERO
              DO 35 I=K,N
                      AUX=DABS(A(I))
                      IF (ETA.LT.AUX) ETA=AUX
35                    CONTINUE
```

```
                 SIGMA=ZERO
                 IF (ETA.EQ.ZERO) GOTO 60
                 KCMPT=0
                 ETA1=ONE/ETA
                 DO 40 I=K,N
                         A(I)=A(I)*ETA1
                         SIGMA=SIGMA+A(I)*A(I)
     40                  CONTINUE
C*****************************************************************
C   APPLY THE TRANSFORMATION AND COMPUTE THE ORTHOGONAL MA
C*****************************************************************
                 SIGMA=DSQRT(SIGMA)
                 IF (A(K).LT.ZERO) SIGMA=-SIGMA
                 A(K)=A(K)+SIGMA
                 BETA=ONE/(SIGMA*A(K))
                 DO 55 I=1,N
                         RO=ZERO
                         DO 45 J=K,N
                                 RO=RO+A(J)*Q(I,J)
     45                          CONTINUE
                         RO=BETA*RO
                         DO 50 J=K,N
                                 Q(I,J)=Q(I,J)-RO*A(J)
     50                          CONTINUE
     55                  CONTINUE
C*****************************************************************
C   COMPUTE THE NEW SUBITERATE.
C*****************************************************************
                 SIGMA=PCNCT/(SIGMA*ETA)
     60          KCMPT=KCMPT+1
                 DO 65 I=1,N
                         Y(I)=Y(I)+SIGMA*Q(I,K)
                         Q(I,K)=HH*Q(I,K)
     65                  CONTINUE
     70          CONTINUE
C*****************************************************************
C   COMPUTE THE NORM OF THE ITERATE AND THE NORM OF THE DI
C BETWEEN THE TWO LAST ITERATES.
C*****************************************************************
         XNORM=ZERO
         DIFIT=ZERO
         DO 75 I=1,N
                 AUX=DABS(Y(I))
                 IF (AUX.GT.XNORM) XNORM=AUX
                 AUX=DABS(X(I)-Y(I))
                 IF (AUX.GT.DIFIT) DIFIT=AUX
                 X(I)=Y(I)
     75          CONTINUE
C*****************************************************************
C   STOPPING CRITERIA.
C*****************************************************************
         IF (KCMPT.GE.N) IER=4
         IF (FNORM.LT.EPS) IER=1
         IF (ITER.GE.ITMAX) IER=2
         IF (DIFIT.LT.(TOL*XNORM)) IER=3
         IF (XNORM.GT.XNMI) IER=5
```

```
      IF (IER.EQ.0) GOTO 10
      ITMAX=ITER
      RETURN
      END
```

Appendix 3

```
C.ROSEN..............................................................*
C                                                                    *
C  FUNCTION          - COMPUTE THE KTH COMPONENT OF ROSENBROCK FUNCTION *
C  PARAMETERS        - K IS THE NUMBER OF THE COMPONENT              *
C                    - X IS THE POINT AT WHICH THE FUNCTION IS EVALUATED *
C  PRECISION         - DOUBLE                                        *
C  LANGUAGE          - FORTRAN                                       *
C                                                                    *
C....................................................................*
      DOUBLE PRECISION FUNCTION ROSEN(K,X)
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION X
      DIMENSION X(2)
      IF (K.GT.1) GOTO 200
100   ROSEN=1.D0-X(1)
      RETURN
200   ROSEN=10.D0*(X(2)-X(1)*X(1))
      RETURN
      END
```

```
C.PSING...............................................................*
C                                                                     *
C   FUNCTION          - COMPUTE THE KTH COMPONENT OF POWELL SINGULAR  *
C                          FUNCTION                                   *
C   PARAMETERS        - K IS THE NUMBER OF THE COMPONENT              *
C                     - X IS THE POINT AT WHICH THE FUNCTION IS EVALUATED *
C   PRECISION         - DOUBLE                                        *
C   LANGUAGE          - FORTRAN                                       *
C                                                                     *
C.....................................................................*
      DOUBLE PRECISION FUNCTION PSING(K,X)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(4)
      IF (K.GT.1) GOTO 200
100   PSING=(X(1)+10.D0*X(2))
      RETURN
200   IF (K.GT.2) GOTO 300
250   PSING=DSQRT(5.D0)*(X(3)-X(4))
      RETURN
300   IF (K.GT.3) GOTO 400
350   PSING=(X(2)-2.D0*X(3))**2
      RETURN
400   PSING=DSQRT(10.D0)*(X(1)-X(4))**2
      RETURN
      END
```

```
C.CHEB.................................................................*
C                                                                      *
C  FUNCTION        - COMPUTE THE KTH COMPONENT OF CHEBYQUAD FUNCTION   *
C  PARAMETERS      - K IS THE NUMBER OF THE COMPONENT                  *
C                  - X IS THE POINT AT WHICH THE FUNCTION IS EVALUATED *
C  PRECISION       - DOUBLE                                            *
C  LANGUAGE        - FORTRAN                                           *
C                                                                      *
C......................................................................*
      DOUBLE PRECISION FUNCTION CHEB(K,X)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(9),TI(9),TIMIN(9),Y(9)
      LOGICAL T
      COMMON N
      CHEB=0.DO
      DO 1 J=1,N
              Y(J)=2.DO*X(J)-1.DO
              CHEB=CHEB-Y(J)
              TI(J)=Y(J)
    1         TIMIN(J)=1.DO
      CHEB=CHEB/N
      IF (K.EQ.1) RETURN
      T=.FALSE.
      DO 3 I=2,K
              T=.NOT.T
              DO 2 J=1,N
                      TIPLUS=2.DO*Y(J)*TI(J)-TIMIN(J)
                      TIMIN(J)=TI(J)
                      TI(J)=TIPLUS
    2                 CONTINUE
    3         CONTINUE
      CHEB=0.DO
      DO 4 J=1,N
    4         CHEB=CHEB-TI(J)
      CHEB=CHEB/N
      IF (T) CHEB=CHEB-1.DO/(K*K-1.DO)
      RETURN
      END
```

```
ISC1.............................................................*
                                                                 *
FUNCTION         - COMPUTE THE KTH COMPONENT OF THE FUNCTION     *
                     OBTAINED BY DISCRETIZATION OF THE TWO POINT *
                     BOUNDARY VALUE PROBLEM                      *
                         U''(T)=DEXP(U(T))       U(0)=U(1)=0     *
PARAMETERS       - K IS THE NUMBER OF THE COMPONENT              *
                 - X IS THE POINT AT WHICH THE FUNCTION IS EVALUATED *
PRECISION        - DOUBLE                                        *
LANGUAGE         - FORTRAN                                       *
                                                                 *
.................................................................*
    DOUBLE PRECISION FUNCTION DISC1(K,X)
    IMPLICIT REAL*8(A-H,O-Z)
    DIMENSION X(1)
    COMMON N
    NP1=N+1
    H2=1.D0/(NP1*NP1)
    IF (K.GT.1) GOTO 200
    DISC1=-2*X(1)+X(2)-H2*DEXP(X(1))
    RETURN
C   IF (K.LT.N) GOTO 300
    DISC1=X(N-1)-2*X(N)-H2*DEXP(X(N))
    RETURN
0   DISC1=X(K-1)-2*X(K)+X(K+1)-H2*DEXP(X(K))
    RETURN
    END
```

```
C.DISC2..............................................................*
C                                                                    *
C  FUNCTION        - COMPUTE THE KTH COMPONENT OF THE FUNCTION      *
C                      OBTAINED BY DISCRETIZATION OF THE TWO POINT   *
C                      BOUNDARY VALUE PROBLEM                        *
C                          U"(T)=(1/2)(U(T)+T+1)**3       U(0)=U(1)=0 *
C  PARAMETERS      - K IS THE NUMBER OF THE COMPONENT               *
C                  - X IS THE POINT AT WHICH THE FUNCTION IS EVALUATED *
C  PRECISION       - DOUBLE                                          *
C  LANGUAGE        - FORTRAN                                         *
C                                                                    *
C....................................................................*
          DOUBLE PRECISION FUNCTION DISC2(K,X)
          IMPLICIT REAL*8(A-H,O-Z)
          DIMENSION X(1)
          COMMON N
          NP1=N+1
          H2=1.D0/(NP1*NP1)
          IF (K.GT.1) GOTO 200
          DISC2=-2*X(1)+X(2)-(H2/2)*((X(1)+1.D0/NP1+1.D0)**3)
          RETURN
   200    IF (K.LT.N) GOTO 300
          DISC2=X(N-1)-2*X(N)-(H2/2)*((X(N)+N/NP1+1.D0)**3)
          RETURN
   300    DISC2=X(K-1)-2*X(K)+X(K+1)-(H2/2)*((X(K)+K/NP1+1.D0)**3)
          RETURN
          END
```

```
C.HAMM1...............................................................*
C                                                                     *
C  FUNCTION        - COMPUTE THE KTH COMPONENT OF THE FUNCTION        *
C                      OBTAINED BY HAMMERSTEIN METHOD APPLIED TO THE  *
C                      TWO POINT BOUNDARY VALUE PROBLEM               *
C                          U"(T)=DEXP(U(T))       U(0)=U(1)=0         *
C  PARAMETERS      - K IS THE NUMBER OF THE COMPONENT                 *
C                  - X IS THE POINT AT WHICH THE FUNCTION IS EVALUATED *
C  PRECISION       - DOUBLE                                           *
C  LANGUAGE        - FORTRAN                                          *
C                                                                     *
C.....................................................................*
      DOUBLE PRECISION FUNCTION HAMM1(K,X)
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON N
      DIMENSION X(1)
      NP1=N+1
      H=1.D0/NP1
      TK=K*H
      HAMM1=0.D0
      DO 1 J=1,K
1              HAMM1=HAMM1+J*DEXP(X(J))
      HAMM1=X(K)+H*(1.D0-TK)*H*HAMM1
      IF (K.EQ.N) RETURN
      KP1=K+1
      SUM=0.D0
      DO 2 J=KP1,N
2              SUM=SUM+(NP1-J)*DEXP(X(J))
      HAMM1=HAMM1+H*TK*SUM*H
      RETURN
      END
```

```
C.HAMM2.............................................................*
C                                                                   *
C  FUNCTION        - COMPUTE THE KTH COMPONENT OF THE FUNCTION      *
C                    OBTAINED BY HAMMERSTEIN METHOD APPLIED TO THE  *
C                    TWO POINT BOUNDARY VALUE PROBLEM               *
C                        U''(T)=(1/2)(U(T)+T+1)**3        U(0)=U(1)=0 *
C  PARAMETERS      - K IS THE NUMBER OF THE COMPONENT               *
C                  - X IS THE POINT AT WHICH THE FUNCTION IS EVALUATED *
C  PRECISION       - DOUBLE                                         *
C  LANGUAGE        - FORTRAN                                        *
C                                                                   *
C...................................................................*
      DOUBLE PRECISION FUNCTION HAMM2(K,X)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(1)
      COMMON N
      NP1=N+1
      H=1.D0/NP1
      TK=K*H
      HAMM2=0.D0
      DO 1 J=1,K
    1       HAMM2=HAMM2+J*0.5D0*(X(J)+J*H+1.D0)**3
      HAMM2=X(K)+H*(1.D0-TK)*H*HAMM2
      IF (K.EQ.N) RETURN
      KP1=K+1
      SUM=0.D0
      DO 2 J=KP1,N
    2       SUM=SUM+(NP1-J)*0.5D0*(X(J)+J*H+1.D0)**3
      HAMM2=HAMM2+H*TK*H*SUM
      RETURN
      END
```

REFERENCES

1. Brent, R. P. (1973), "Some Efficient Algorithms for Solving Systems of Nonlinear Equations," SIAM Journal Numerical Analysis 10, Number 2, pp. 327-366.

2. Brown, K. M., and Dennis J. E., Jr. (1971), "On the Second Order Convergence of Brown's Derivative-free Method for Solving Simultaneous Nonlinear Equations," Yale University, Computer Science Research Report 71-7.

3. Brown, K. M. (1968), "A Quadratically Convergent Newton-like Method Based upon Gaussian Elimination," Cornell University Computer Science Technical Report 68-23.

4. Broyden, C. G. (1965), "A Class of Methods for Solving Nonlinear Simultaneous Equations," Maths. Computation Volume 19, pp. 577-593.

5. Broyden, C. G. (1971), "The convergence of an Algorithm for Solving Sparse Nonlinear Systems," Maths. Computation, Volume 25, pp. 285-294.

6. Dennis, J. E., Jr. and Moré, J. J. (1974), "A Characterization of Superlinear Convergence and its Application to Quasi-Newton Methods," Maths. Computation Volume 28, pp. 549-560.

7. Fletcher, R. (1965), "Function Minimization Without Evaluating Derivatives - A Review," Computer Journal Volume 8, pp. 33-41.

8. Gay, D. M. (1975), "Brown's Method and Some Generalizations with Applications to Minimization Problems," Ph.D. Thesis, Cornell University, Computer Science Technical Report 75-225.

9. Gentleman, W. N. (1973), "Least Squares Computations by Givens Transformations Without Square Roots," Journal Inst. Maths. Applics. 12, pp. 329-336.

10. Gill, P. E., Golub, G. H., Murray, W., and Saunders, M. A. (1974), "Some Methods for Modifying Matrix Factorizations," Mathematics of Computation

11. Gragg, W. B. and Stewart, G. W. (1974), "A Stable Variant of the Secant Method for Solving Nonlinear Equations," (manuscript).

12. Hanson, R. J. (1973), "Least Squares Computation with Two Algorithms for the Two-Multiply, Two-Add Givens Rotation," Washington State University Computer Science Department, Technical Report.

13. Moré, J. J. and Trangenstein, J. (1974), "On the Local Convergence of Broyden's Method," Cornell University Computer Science Technical Report 74-216.

14. Ortega, J. M. and Rheinbolt, W. C. (1970), Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, New York.

15. Powell, M. J. D. (1970a), "A Hybrid Method for Nonlinear Equations," in Numerical Methods for Nonlinear Equations edited by P. Rabinowitz, Gordon-Breach, New York.

16. Powell, M. J. D. (1970b), "A Fortran Subroutine for Solving Systems of Nonlinear Algebraic Equations," in Numerical Methods for Nonlinear Equations edited by P. Rabinowitz, Gordon-Breach, New York.

17. Stewart, G. W. (1974), Introduction to Matrix Computation, Academic Press, New York.